A-Shell

XCALL Reference

Copyright © 2005 Jack McGregor. All rights reserved.
This edition was produced on 1 September 2005.

Publisher

MicroSabio
22704 Ventura Blvd., #476
Woodland Hills, CA 91364 USA
Web site: www.microsabio.com
Email: info@microsabio.com
voice: 818-710-8437
fax: 818-704-4882

Authors

Jack McGregor has been developing and documenting A-Shell since 1994, and his authorship continues through the date of this publication. Ty Griffin is responsible for editing, formating and publishing this document in its various forms.

Related Information

This document contains the most up-to-date information that was available at the time of publication. It is part of the overall A-Shell documentation set, which consists of the following major components:

- Set-Up Guide
- Command Reference
- Development Guide
- XCALL Reference
- Update Notes

Other related materials include the *A-Shell Telnet Service Guide*, the *A-Shell COM Interface Reference*, and the *Ashlite Program Reference*. All of these publications, in various formats, are available from the MicroSabio web site documents page. Additional information about A-Shell may be included in files that accompany the distributed (i.e., installation) copies of the software.

The documents in the A-Shell collection are available in multiple formats: HTML, Windows Help (CHM) and PDF, to name the most common. If you do not easily find the document format you prefer, please send a request for that format to MicroSabio.

Legal Notice

This documentation and the software it describes contain proprietary information belonging to MicroSabio, a California proprietorship owned entirely by Jack McGregor. The software and this related information is provided under a license agreement containing restrictions on use and disclosure, and is protected by copyright law. This information is confidential between MicroSabio and the client, and remains the exclusive property of MicroSabio.

Due to continued product development, the information contained herein may change without notice. It is believed to be accurate and reliable, at least at the time of its writing. However, no responsibility for the accuracy, completeness or use of this information is assumed by MicroSabio. If you find any problems in the documentation, please report them to the publisher.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the expressed or licensed permission of MicroSabio.

Trademarks

MicroSabio, A-Shell, INMEMO, INFLD and EZ-SPOOL are trademarks of Jack McGregor.

AIX and Informix are registered trademarks of International Business Machines Corporation.
Alpha Micro, AMOS, and AlphaBASIC are registered trademarks of Alpha Micro Products, Inc.
HP-UX is a trademark of the Hewlett-Packard Company.
Linux is a registered trademark of Linus Torvalds.
Microsoft and Windows are registered trademarks of Microsoft Corporation.
Red Hat is a trademark of Red Hat, Inc.
UNIX is a registered trademark of The Open Group.
ZTERM is a trademark of Rod Hewitt at Cool.stf.

All other products, services, companies, events and publications are trademarks, registered trademarks or service marks of their respective owners.

Contents

Updates 7

Introduction 9
- Creating Routines 10
- Configurable Routines 10

Alphabetical List of Routines 11

Routines by Function 16

Detailed Descriptions 18

Appendix 347
- Virtual Key Symbolic Names 348

Index 349

Updates

Listed in the table below are descriptions of the changes to the documentation and A-Shell that have occurred since the release of A-Shell 4.9 in October 2003. This table is provided so long-time users can find recent changes; if you're new to A-Shell, just ignore these update notes.

The changes are listed in reverse chronological order. The "build" number is given so you can determine if the version of A-Shell you are running (see the *Help* menu) includes the capabilities and changes described below.

The AUI and PCKLST/XTREE routines are described in this document, but their extensive changes are omitted from following table.

Build, Date	Topic	Description
936, 9 Aug 05	MIAMEX 65	Strip trailing blanks
936, 8 Aug 05	IMAGE	Add support for millirows
932, 31 May 05	MIAMEX 148	Calculate string length or height
931, 22 May 05	MIAMEX 147	Count lines in file
930, 19 May 05	MIAMEX 146	Get last line number
925, 30 Mar 05	MIAMEX 138	Add option to replace null delimiters
n/a	MIAMEX 60	Add GOP2'xxx table of values
n/a	BASORT	Add note about more powerful (Opttech) sort
916, 31 Jan 05	MIAMEX 108	Add option to load a module into user memory
915, 27 Jan 05	TRMCHR	Add WINROW and WINCOL
909, 19 Dec 04	MIAMEX 141	Set parent control
908, 17 Nov 04	TCPX	Major enhancement
908, 17 Nov 04	MIAMEX 138	Registry operations
904, 18 Oct 04	MIAMEX 72	Add “default message” parameter
904, 18 Oct 04	CGIUTL	Add Opcode 6
n/a	STRTOK	Add two "real world" examples to doc
899, 7 Sep 04	MIAMEX 118	Add filidx parameter
897, 18 Aug 04	Decimal ppns	GETJTB, GETJOB, GETPPN, MIAMEX 11, MIAMEX 119
893, 24 Jun 04	MIAMEX 133	Expand a file in place
892, 10 Jun 04	ERRMSG	Displays Windows-style message box

Build, Date	Topic	Description
889, 25 May 04	EFS	Encrypted file service
885, 03 May 04	MIAMEX 132	Reformat filetime from MIAMEX'131
885, 03 May 04	MIAMEX 131	Retrieve stats for specified path
884, 01 May 04	MIAMEX 130	Retrieve startup command
884, 28 Apr 04	MSBOXX	Improve operation of pop-up box
881, 14 Apr 04	MIAMEX 129	Get or set GUI-related flags
879, 12 Apr 04	MIAMEX 128	Get IP address
878, 02 Apr 04	MIAMEX 127	Get or set rounding factor
875, 11 Mar 04	MIAMEX 60	Add more options via second parameter
866, 12 Feb 04	MIAMEX 126	Sink or unsink specified box
863, 02 Feb 04	MIAMEX 91	Link screen colors to Window system colors
863, 02 Feb 04	MSBOXX	Displays GUI-style pop-up window
862, 02 Feb 04	MIAMEX 125	Retrieve info about last mouse click
862, 02 Feb 04	MIAMEX 124	Write STRING message to ashlog.log
861, 21 Jan 04	MIAMEX 119	Create static text controls
858, 12 Jan 04	SORTIT	Sorts up to six sort keys
855, 28 Nov 03	TCPX	Socket communications
854, 21 Nov 03	STRTOK	Parse strings
854, 21 Nov 03	MIAMEX 71	Permits specifying "virtual keys"
854, 21 Nov 03	MIAMEX 107	Maximum modules per job now 96
852, 06 Oct 03	AMOS	Supports "process control" flag
851, 05 Oct 03	ASFLAG	Memory mapping may be turned off
850, 04 Oct 03	MSGBOX	Supports a HELP button

Introduction

AlphaBASIC programs rely heavily on the use of XCALL subroutines to perform functions from the simple **STRIP** (strip trailing spaces from a string) to the more complex **BASORT** (sort sequential and random data files). XCALL subroutines under AMOS have the extension SBR, and traditionally live in DSK0:[7,6] (BAS:). When a program calls a subroutine, AlphaBASIC locates the SBR file, loads it if necessary, and then calls to the start of it.

XCALL subroutine handling under A-Shell is functionally identical: your program invokes a routine, and A-Shell calls to its start. Because of the structural differences between AMOS/AlphaBASIC and A-Shell, however, A-Shell subroutines do not (normally) exist as separate SBR files; they are, instead, linked into the A-Shell object module and are therefore always available.

A-Shell comes with over 200 subroutines, including nearly all of the so-called AlphaAccounting routines from [7,60] and the so-called "standard AlphaBASIC subroutines" from [7,6]. It also includes the most popular MicroSabio subroutines for AMOS, (INFLD, INMEMO, EZSPL, MSBOXX), plus a variety of routines that have been written specifically for A-Shell. In addition, A-Shell includes many third-party subroutines that have been converted over the years as more firms and developers have migrated to A-Shell and we have converted their AMOS routines.

This document lists about 150 of the most commonly used routines, and provides detailed documentation for most of them. Those not listed were skipped because they seemed too obscure, are specific to individual users, or because they have been too recently added—a process which goes on more or less continuously. If there is a function that you need and do not see, be sure to inquire about it before writing your own.

Creating Routines

A-Shell also supports external, dynamically-loaded subroutines which you can write in BASIC and which are called in the exact same way from within your program. These routines have an extension of **Error! Reference source not found.**, so named to avoid confusion with the SBR extension used under AMOS. Information on writing your own routines is included in the *A-Shell Development Guide*.

Configurable Routines

Some of the routines have been modified by developers for their own purposes. In order to make those changes and improvements available to all A-Shell users, many of the more common and useful modifications have been implemented as configurable patches. These are specified using the **SBR** command in the configuration file MIAME.INI. Full details of this command, and the subroutine patches, are given in the *A-Shell Setup Guide.* The subroutines which have configurable functionality are:

INFLD	MESAG	COMMON	BASORT	XLOCK	ODTIM
SERCH	LSTLIN	JOBNAM	PRINT	TRIM	PRTCHK
XPPN	AMOS				

The subroutines GETJTB, SETJTB and PRTCHK are supplied for use under AMOS in the AMOS subdirectory. It is hoped that this will ease the writing of portable code. In particular, use of GETJTB can eliminate the need for many of the GETxxx routines to get information about your job, terminal, ppn, etc.

Alphabetical List of Routines

The following table lists A-Shell's most generally useful routines. Most of these routines are embedded in A-Shell, but some are written in BASIC and are provided as separate files. In the "Source" column abbreviations are used for AlphaAccounting ("AA"), AlphaAccounting UK ("AA UK") and MicroSabio ("MS"). Most of these routines, including all of the commonly-used ones, are described in significant detail in the "Detailed Descriptions" section.

Routine	Origin	Function
ABOX	Unknown	Draw a box
ACCEPN	AlphaBASIC	Input single character (F6 or S1) with no echo
ACCEPT	AlphaBASIC	Input single character with echo
ACCESS	Misc	Display one-line menu, accept one-char input
AMBTOA	AM Belgium	Miscellaneous functions including timed input
AMOS	MS	Execute AMOS command
ANYCN	AA / MS	Input "Any change?" number
ASCEBC	Unknown	ASCII-EBCDIC conversion
ASFLAG	MS	Set A-Shell runtime flags: read-only,syncwrite,etc.
AUI	MS	A-Shell's GUI interface and toolkit
AUTLOG	Soft Machines	Interface to AutoLog
B64ENC	A-Shell	Base 64 encoding (for email attachments)
BASORT	AlphaBASIC	Sort sequential and random data files
BITOPS	HMOpro	Bit field operations
BLOCKS	AA UK	Return number of free disk blocks
BOX1C	Debug plc	Various box drawing functions
BUTTED	Unknown	*xcall butted, string* removes all but 0-9 and A-Z
CCOFF	AA UK	Disable Control-C interrupts
CCON	AA UK	Enable Control-C interrupts
CGIUTL	A-Shell	CGI programming utilities
CHKKBD	Misc	Check kbd; optionally input char if available: *xcall chkkbd, flag (B,1), {char (S,1)}*

Routine	Origin	Function
CHKONE	Unknown	Check kbd to see if char is available
CHKSPL	Unknown	Checks if spooler exists; must use ALIAS =CHKSPL:PRTCHK; similar to PRTCHK
CMDR	Debug plc	Put command file in :R mode
COMIO	MS	Serial port I/O under Windows
COMMON	AlphaBASIC	Inter-program communication; configurable
CONDEV	A-Shell	Get IP address, console device or machine name
CRC16	ERS	Calculate 16 bit CRC on block of data
CREMLX	ADGAP	Build multi-level index used by SEARCH
DATES	Unknown	Many date conversions, Julian, etc.
DELCHR	A-Shell	Remove specified characters from string
DEVCHK	AA UK	Check for valid device specification
DSKPPN	Unknown	Return disk and ppn
DSPLY	AA	Display variables with various formats
DSTOI	A-Shell	Convert separated date to Julian
EBCASC	Unknown	ASCII-EBCDIC conversion
ECHO	AlphaBASIC	Enter line mode. Turn on terminal echo
EMAILP	MS	Email a report
ERRMSG	MS	Error trap reporting routine; must alias to DERR
EZSPL	MS	Send file to printer with enhanced options
EZTYP	MS	Display file with paging, 80/132 switching
F2HOST	A-Shell	Convert F,6 to IEEE format
FIFO	MS	Named pipe communications under UNIX
FILL	Unknown	Variation of AlphaBASIC Plus FILL$() function: *xcall fill, var, pattern {,length}*
FILNAM	Unknown	*xcall filnam, ch, fspec* returns the file spec for the specified open (or attempted) file channel.
FILOCK	AA UK	Front-end to FLOCK.SBR
FINB	Debug plc	Input 1 byte from file*: xcall finb, chan, byte, status*
FLOCK	AlphaBASIC	File/record locking control
FOLD	Various	Various upper/lower case folding operations
FORCE	A-Shell	Force characters into another job's input
FUNKEY	MS	Display function key line message
GDIPRT	MS	Send report to AshLite client for printing
GET	AlphaSoft	Get single character from keyboard or file
GETADR	MS	Return address and size of variable
GETBYT	Unknown	Read raw bytes from a sequential file
GETDEV	AA UK	Get current device
GETDSK	AA UK	Get current device

Routine	Origin	Function
GETJOB	AA UK	Get job name, program, ppn, job number
GETJTB	Debug plc	Get job table information
GETLOG	Unknown	Get jobname, number, and ppn; must use *alias =getlog:getjob*
GETMAC	MS	Get MAC (hardware) address of Ethernet controller
GETPPN	AA UK	Get current PPN
GETPRG	MS	Get current program or SBX name
GETTRM	MS	Get current terminal name: *xcall gettrm, trmdef$*
GETUSN	Custom	Get user login name
GETVER	MS	Get current program version
GETX	A-Shell	Get single character with function key translation
GRECNV	SSCI	Various Julian date manipulations
GTJBNO	A-Shell	*Xcall gtjbno, jobnam, jobno* returns job number for specified job name (or 0 for error)
GTLANG	AlphaBASIC	Return current language information
HOST2F	A-Shell	Convert IEEE 4 or 8 byte float to BASIC 6 byte
HOSTEX	A-Shell	Execute host operating system cmd as a subroutine
HTLMP	MS	View report in browser
IDTIM	AlphaBASIC	Input date
IMAGE	A-Shell	Display images (BMP,PCX,JPG,TIF format)
INCOM	MS	Variation of COMMON
INFLD	MS	Extended AlphaAccounting input routine
INMEMO	MS	Memo/Jotter editing/handling routine
INPUT	UK	Identical to INFLD
INPUTC	AA UK	Input variation used in the UK
INVUE	Foxware	Requires ALIAS=INVUE:INFLD
ISMROK	DMSI	Return information from ISAM 1.0 “rock”
ITC	MegaSoft	(UNIX only) Send message to another process
JOBCMD	MS	Set command to be executed whenever at command level
JOBDAT	inSight	Get information about your job
JOBNAM	Unknown	Equivalent to GETJOB
JOBNUM	AA UK	Return unique job number
JOBTRM	Unknown	Return jobnam, trmdef, tdvnam
JULCVT	Unknown	Julian date conversions
KDAY	AA UK	Calculate elapsed days to given date (b,2)
LAPSED	AA UK	Calculate elapsed days to given date (f,6)
LOG	MS	Log to a new PPN or retrieve PPN info.
LOGRIO	MS	Logical record i/o (any size records)
LSTLIN	A-Shell	Return last command line
MATCH	HMOpro	Compares a data field to a list of values or ranges

Routine	Origin	Function
MESAG	AA	Display message on bottom line of screen
MIAMEX	A-Shell	Over 100 A-Shell interface functions
MMENU	AA	Display file maintenance menu
MOUNT	AA	Display device mount message
MPSCOM	Custom	Interface to MacDonald PartSelect catalog
MSBOXX	MS	Box and window utility
MSGBOX	MS	Displays a message in a dialog box format
MSGLOG	AlphaBASIC	Output coded message to SYSLOG.SYS
NFIND	AA UK	Perform string search forwards or backwards
NOECHO	AlphaBASIC	Enter image mode. Turn off terminal echo
NOEKO	MS	Same as NOECHO
ODTIM	AlphaBASIC	Output formatted time and date
PACK	A-Shell	Pack 3 char string to B,2 RAD50*: xcall pack, str, rad*
PCKLST	HMOpro	Display a pop-up pick list.
PGMID	MS	Display screen header with program name, time, etc
PGMND	AA	Display end of program message
PLYJOB	MS	Return detailed job info about current or other job
PPNSWP	Unknown	Log to a new device:[p,pn]. Must use ALIAS= PPNSWP:SETPPN
PRINT	AA	Build formatted print file
PRIV	AA	Dummy routine (not necessary to log to 1,2 here)
PRTCHK	AA UK	Check printer name and printer queue
PUTBYT	Unknown	Write raw bytes to a sequential file
RDATE	AA	Get system date
RENAM	MS	Rename a file
RENAME	AlphaBASIC	Rename a file
ROUND	Custom	Round argument to nearest integer
RVCOMN	Custom	Like COMMON.SBR with one 1024 byte packet.
RXTERM	MS	Check input buffer for function key (Windows only)
SBRC	UK	Misc functions used by the R/W and P/G packages
SCGINP	Unknown	Variation of INPUT.SBR
SCRN	Willowtree	Willowtree Screen Handler
SEARCH	ADGAP	Enhanced (disk optimized) variation of SERCH
SEND	A-Shell	Send a message to another terminal
SERCH	AA	Search random data file
SETDEV	AA UK	Change current device: *xcall setdev, « dsk2 »*
SETJTB	Debug plc	Set job table information
SETPPN	AA UK	Change current PPN: *xcall setppn,b2*
SIZE	A-Shell	Returns size of specified file in bytes

Routine	Origin	Function
SLEEP	AlphaBASIC	Stall processing for number of seconds
SORTIT	MegaSoft	Sort array in memory
SPOOL	AlphaBASIC	Send file to printer
SRCH2	AA	Search random data file
STALL	AA	Stall processing for number of seconds
STENO	AA	Input starting/ending numbers
STIME	Dalcon	Return current time in formatted string
STRIP	AlphaBASIC	Strip trailing spaces from string
STRTOK	MS	Parse strings
SUBMIT	Custom	Launch a background task (UNIX only)
SWPSBR	Swap	Various Swap-related functions
SYS000	Misc	Similar to COMMON (one packet, passive read): *xcall sys000, sr (b,1), packet (x,100)*
TBOX	Misc	Draw box around specified SR,SC,ER,EC
TCPCLI	MS	Socket communications (client interface)
TCPSRV	MS	Socket communications (server interface)
TCPX	MS	Socket communications
TINKEY	AlphaSoft	Check for terminal input
TMEN2	AA	Display reduced transaction menu
TMENU	AA	Display transaction menu
TRIM	Debug plc	Strip trailing and leading spaces from string
TRMCHR	AlphaBASIC	Return terminal characteristics
UNIQUE	Unknown	Generate unique temp filename
UNPACK	AA UK	Unpack RAD50 word, e.g. *xcall unpack,b2,s3*
USRCNT	MS	Get current user count & number of nodes licensed
USRTBL	Custom	*xcall usrtbl, name, code, op*
VUESCR	Swap	Edit a rectangular array of text
WAIT	AA	Output ‘Please wait....’ messages
WAKNO	MS	(UNIX only) Wake up a sleeping job
WHOAMI	AA UK	Get job name
WINFLG	inSight	Get inSight information about your job.
XDEFLT	MS	Insert text into the type-ahead buffer
XFOLD	HMOpro	Intelligent folding of names
XFRMMO	MS	Move or copy an INMEMO logical memo
XLOCK	AlphaBASIC	Shared resource control
XMOUNT	AlphaBASIC	Mount a device. Dummy routine
XPAINT	MS	“Execute” an AlphaPAINT 2.0 screen display
XPPN	Unknown	Get ppn, device, job, term
XTREE	MS	Windows "tree" control

Routines by Function

Graphical User Inferface

AUI	GDIPRT	HTLMP	IMAGE	INFLD	MESAG
MSBOXX	MSGBOX	PCKLST	XTREE		

Keyboard Operations

ACCEPN	ACCEPT	ACCESS	AMBTOA	ANYCN	CHKKBD
CHKONE	GET	GETX	INFLD	INMEMO	INPUT
INPUTC	INVUE	NOECHO	NOEKO	PCKLST	RXTERM
SCGINP	STENO	TINKEY	TMEN2	TMENU	XDEFLT

Disk Operations (File I/O)

BASORT	BLOCKS	CGIUTL	CREMLX	ERRMSG	EZSPL
EZTYP	FIFO	FILNAM	FILOCKFINB	FINB	FLOCK
GET	GETBYT	IMAGE	INMEMO	ISMROK	LOGRIO
MSGLOG	PCKLST	PRINT	PUTBYT	RENAM	RENAME
SEARCH	SERCH	SIZE	SRCH2	UNIQUE	XFRMMO

Job Environment

AMOS	CCOFF	CCON	CMDR	CONDEV	DSKPPN
ECHO	ERRMSG	FILNAM	FORCE	GETDEV	GETDSK
GETJOB	GETJTB	GETLOG	GETMAC	GETPPN	GETPRG
GETTRM	GETUSN	GETVER	GTJBNO	GTLANG	JOBCMD
JOBDAT	JOBNAM	JOBNUM	JOBTRM	LOG	LSTLIN
PLYJOB	PPNSWP	PRIV	SETDEV	SETJTB	SETPPN
TRMCHR	WHOAMI	WINFLG	XPPN		

System

AMOS	ASFLAG	BLOCKS	CGIUTL	DEVCHK	EMAILP
EZSPL	FIFO	FORCE	HOSTEX	USRCNT	

Data

ASCEBC	B64ENC	BASORT	BITOPS	BUTTED	CRC16
DELCHR	EBCASC	F2HOST	FILL	FILNAM	FOLD
HOST2F	MATCH	NFIND	PACK	ROUND	SORTIT
STRIP	STRTOK	TRIM	UNPACK	XFOLD	

Display / Screen

ABOX	BOX1C	DSPLY	ERRMSG	EZTYP	FUNKEY
IMAGE	INFLD	INMEMO	MESAG	MMENU	MOUNT
MSBOXX	MSGBOX	PCKLST	PGMID	PGMND	SCRN
STENO	TBOX	TMEN2	TMENU	VUESCR	WAIT
XPAINT					

Networking

CGIUTL	CONDEV	EMAILP	GDIPRT	GETMAC	HTLMP
TCPX					

Dates

DATES	DSTOI	GRECNV	IDTIM	JULCVT	KDAY
LAPSED	ODTIM	STIME			

Printing

CHKSPL	EMAILP	EZSPL	EZTYP	GDIPRT	HTLMP
PRTCHK	RDATE	SPOOL			

Interjob Communications and Control

FLOCK	FORCE	INCOM	ITC	PLYJOB	RVCOMN
SEND	XLOCK				

Detailed Descriptions

On the following pages are in-depth descriptions of each routine.

ABOX

```
xcall ABOX, srow, scol, height, width {,opcode}
```

ABOX.SBR is one of the many routines which draw boxes (among which, MSBOXX would be the most sophisticated and preferred from our standpoint). Note that this version draws the box *at* the specified position (rather than *around* it) and that the third and fourth parameters are height and width rather than the ending row and column. The default action is to just draw the box border; if *opcode* is specified and non-zero, the interior of the box will also be cleared.

ACCEPT, ACCEPN

```
xcall ACCEPT, key {,xlt}

xcall ACCEPN, key {,xlt}
```

ACCEPT.SBR is an enhanced version of the AlphaBASIC subroutine of the same name which inputs a single character from the keyboard. One of the enhancements is that the *key* parameter, in addition to the normal floating point type, can also be a one byte string. Two additional enhancements relate to the optional *xlt* parameter, which must be of type String. If you pass it as a three-character string, such as "IFX", then the routine will translate any input sequences using the function key translation table DSK0:<tdvname>.<XLT string>[7,0]. For example if *xlt* is "IFX" and the current terminal driver is "AM62A", then it would use DSK0:AM62A.IFX[7,0]. If you pass *xlt* as a null string, then there will be no function key translations, but it will still use the "cooked" input routine, which operates at a higher level than the "raw" routine used when *xlt* is not specified at all. The "cooked" routine, for example, will return characters left over from prior function key translations. In no case, though, will **ACCEPT** (or **ACCEPN**) return input from a command file.

ACCEPN.SBR is the same as **ACCEPT.SBR**, except that it does not echo the character input. **ACCEPT** will echo the character, unless it is an unprintable character.

ACCESS

```
xcall ACCESS, mnustr, op$, extflg {,row {,col}}
```

ACCESS.SBR is a handy utility for implementing single-line, single-character selection menus. The menu is made up of one or more choices, where each choice is displayed as a single upper case character followed by a close parenthesis and one or more lower case characters. The upper case characters are used to make the selections, and must be unique. A typical menu might look something like this:

```
A)dd C)hange D)elete Q)uit
```

Parameters

mnustr (null terminated string of appropriate length)

must be formatted exactly as displayed (see example above) except that the spaces between the items are optional. Also, the string may begin with one or more of the following shortcut letters (with no following parentheses):

Shortcut	Displays as
A	A)dd
C	C)hange
D	D)elete
S	S)can
I	I)nquiry
X	X)ternal
L	L)ist

For example, if MNUSTR = "ACDLP)ostQ)uit" the menu would appear as:

```
A)dd C)hange D)elete L)ist P)ost Q)uit
```

op$ (S,1)

may be loaded with the character corresponding to the default option, and will be returned with the character entered by the user to make a selection. (The user must enter one of the upper case characters in the list, or use BACKSPACE, ESC, or a function key to exit.)

extflg (F,6)

will return 0 if a normal menu selection was made, or 1 if BACKSPACE or ESC was hit, or a number in the range of negative 1 to negative 32 if a function key is hit. In order for the function keys to be accepted, you must have a <tdvname>.IFX file in the LIB: account which maps function keys according to the INFLD scheme (e.g. F1 = ^G^A, F2 = ^G^B, etc.).

row, col

may be optionally used to specify the starting position to display the menu. If ROW is 0 or not specified, the menu will be displayed starting from the position of the cursor at the time of the XCALL.

AMOS

Documentation update June 04; see "Update Notes" below.

```
xcall AMOS, command {,"Q"}
```

AMOS.SBR allows a program to execute a command (e.g. DIR, LOG, COPY, etc.) or BASIC program (e.g. RUN XYZ) almost as if it were a subroutine. It is similar to HOSTEX except that instead of executing host operating system commands (which differ from one operating system to the next) it executes AMOS commands, which are emulated by A-Shell and behave identically among A-Shell's different platforms.

AMOS.SBR may work in one of three ways, depending on whether the SBR=AMOS_RUNSBR switch is set, and depending on the command line specified.

- If SBR=AMOS_RUNSBR is not set, or if the command does not begin with "RUN" or "ORUN", and it is not one of the special cases listed below, then it works by spawning a new instance of A-Shell with the *exit-upon-completion* switch, forcing the new instance to execute the desired command. The first instance is suspended until the second instance completes the command. This is essentially the same as using HOSTEX and passing it a command line starting with the A-Shell executable name. Because the new instance requires nearly the same amount` of memory as the first instance, this subroutine is only supported under A-Shell/Windows and A-Shell/UNIX, both of which can draw on virtual memory.
- If the SBR=AMOS_RUNSBR switch is set, and the command begins with "RUN" or "ORUN", then the command will be executed as a true subroutine with the current process and job. The current program is suspended, and a new memory partition is allocated for the subroutine, but it shares the original job's job table and process environment. Any screen output, and any changes to the job environment will remain when the subroutine exits. See the discussion on the AMOS_RUNSBR switch in the *A-Shell Setup Guide* for important information on the limitations and tricks related to this switch.
- If a second instance is going to be launched, then if the optional second argument is specified, an attempt is made to execute the second instance "quietly". Under A-Shell/Windows, this is accomplished by making the new instance run in minimized mode. Under UNIX, it uses the –q startup switch.

In most cases, the A-Shell implementation of **AMOS.SBR** will achieve the same result as the typical implementation under AMOS. Actually, there are several versions of this subroutine created and distributed by various parties in the AMOS community. But most, if not all, work by making the child instance run as if it were a subroutine called by the parent. If you do not set the AMOS_RUNSBR switch, or if the command cannot otherwise be run within the current process, then the fact of it running in a separate process can lead to some subtle differences in the outcome which may be significant to some applications. For example, if the purpose of the child process is to make some change to the job status (e.g. via the SET command) then any such change will be discarded when the subroutine returns if the subroutine ran in a separate process. Another example would be where the child process uses its job name to generate a file or key, assuming that its job name is the same as the parent.

To minimize the chance of misunderstanding, the A-Shell implementation of AMOS tries to be smart by looking at the command and deciding if it is possible or more desirable to execute that command *in-process* (as a true subroutine) rather than by launching a new process. Some examples of this are:

```
xcall AMOS,"LOG <dev:[p,pn]>"
```

Since this is clearly a case where the intent is to change the PPN of the calling job, A-Shell intercepts this and turns it into XCALL LOG,"<dev:[p,pn]>."

```
xcall AMOS,"LOOK <fspec>"
```

This one could be handled as a separate process with little harm, but since the new process would just turn around and use XCALL EZTYP,<fspec> (MicroSabio's EZTYP utility being deemed the appropriate replacement for Micro Concept's LOOK.LIT) it seems more efficient to just call it directly.

```
xcall AMOS,"VUE <fspec>"
```

The reasoning here is similar to that used above. Under A-Shell, **VUE.LIT** is actually just a front end to the VUE function which is embedded within the A-Shell executable, so it makes just as much sense to call that function within the current instance as it does to start a new one.

Comments

AMOS.SBR will launch the new instance of A-Shell (if necessary) using the same initialization file and executable command pathname with which the current instance was launched.

The child job can determine its parent jobname from the JOBATT parameter of either GETJTB or PLYJOB.

The configuration file option switch, SBR=AMOSJOB1, will cause **JOBNAM.SBR** to return the name of the parent job rather than the child job.

Despite the name AMOS, you are still limited to executing commands that are recognized by and compatible with A-Shell. You cannot, for example, take AMOS machine language executables and execute them this way. (The A-Shell LIT commands have been rewritten to emulate their AMOS counterparts, but have a completely different internal format.)

Update Notes: Build 852.1 - 06 Oct 03

AMOS.SBR now supports an optional 3rd parameter: a "process control" flag. The new syntax is:

```
xcall AMOS, cmd {,qflag {,pflag}}
```

If *qflag* is non-zero, non-blank, it causes the subroutine to run "quietly" (generally invisibly). If *pflag* is 1, it forces subroutine to run as if the AMOSRUNSBR flag had been set. If *pflag* is 2, it forces the subroutine to run as if AMOSRUNSBR had NOT been set. Any other value causes the normal AMOSRUNSBR setting as specified in the MIAME.INI file or **SET.LIT** command to prevail.

ANYCN

```
xcall ANYCN, cngctl, whatno
```

ANYCN.SBR is used in many AlphaAccounting-derived programs to ask if the operator wants to make any changes before updating a record, and if so, which field number. The A-Shell implementation of **ANYCN** is pretty faithful to the AMOS one, except for a couple of extensions: support of foreign languages via the SBRMSG.lan file, and support of function key response codes. For convenience a complete summary of the functionality follows.

The *cngctl* parameter is used to control certain variations of the function as described below:

Value	Meaning
1	Display the message "Any Change?" on line 24; allow the input of affirmative, negative, or a number. If an affirmative response given, the operator is then prompted for "What Number?". If a number is entered, it is treated as if an affirmative response was first given and then the number was entered at the "What Number?" prompt.
2	Same as for 1, except that there is no option to input a number – only negative and affirmative responses are allowed.
3	Display the message "INVALID SELECTION" and then proceed to prompt as if CNGCTL was set to 1.

On return from the routine, *cngctl* will be set to –1 if ESCAPE was entered, 0 for a negative response and 1 for an affirmative response. In the last case, *whatno* will be set to the number entered. If a function key is entered and the <tdv>.IFX function key translation file has been set up using the INFLD standard for function keys (i.e. F1 = ^g^a, F2 = ^g^b, etc.) then the negative of the function key number will be returned in *whatno* (e.g. F1 returns –1, F2 returns –2, etc.).

ASCEBC, EBCASC

```
xcall ASCEBC, record

xcall EBCASC, record
```

ASCEBC.SBR and **EBCASC.SBR** convert the specified record from ASCII to EBCDIC and from EBCDIC to ASCII, respectively. The single parameter is a string or unformatted record of any size, which will be converted in place.

ASFLAG

Documentation update June 04; see "Update Notes" below.

```
xcall ASFLAG, flag
```

ASFLAG.SBR allows a program to set various internal A-Shell options that do not fall conveniently under another category. The *flag* argument is treated as a bitmap whose currently defined options are as follows.

Value	Meaning
	Same as 1 if omitted (i.e. XCALL ASFLAG equals XCALL ASFLAG,1)
0	Turn off all flags
1	Turn on Read Only
2	Turn on Synchronized Write mode for D-ISAM
3	Turn on both the Read Only and Synchronize Write flags
4	Allow Divide By 0
16	Memory Mapping
32	No IDX Lock
64	Make Local Copy of file (Windows only)
128	May be used within an **Error! Reference source not found.** subroutine to force a Force Control-C to Parent program (on return from the SBX)
256	Turn off memory mapping

Comments

This subroutine was first introduced as SETRO since its only function was the "Set Read Only" mode. **ASFLAG** is backward compatible with SETRO, so you can either change the name in your source code or alias it in MIAME.INI (e.g. ALIAS=SETRO:ASFLAG).

A technique related to the memory mapping and local file schemes described above for speeding up single-user or read-only access to a file is that of loading a file into user memory (with **LOAD.LIT** or MIAMEX 108) and then accessing it via the MEM: device. Refer to the *A-Shell Development Guide* for more details about that.

Update Notes: Build 851.2 - 05 Oct 03

Flag 256 turns off memory mapping (overriding the MMAPLIST in the MIAME.INI) for any files subsequently opened in the current program.

Read Only

This mode causes any files opened subsequently for Random'Forced access within the current program to be treated as if the Read'Only open flag had been specified. (Since Read'Only was not introduced into the AMOS version of AlphaBASIC until late in the game, some developers use **ASFLAG.SBR** instead to

maintain source code compatibility with early versions of the AlphaBASIC compiler, supplying a dummy **ASFLAG.SBR** under AMOS.)

The motivation for Read'Only under AMOS was strictly to avoid file locking conflicts in a program that was merely reading records and did not care whether they were locked (such as in a report). Under A-Shell/UNIX, this is the default behavior anyway, so the original motivation is only applicable to Windows platforms. However, there is another motivation for this mode, which applies to all networked A-Shell platforms: it can dramatically speed up file access, even if there are no other users accessing the same file. The reason has to do with eliminating the need for communication between the client process and the locking daemon on the server, which is especially slow under NFS and some Windows networks. Experimenting with this is relatively simple, since you can always "remove" the ASFLAG calls by simply aliasing ASFLAG to a null subroutine, like PRIV (e.g. ALIAS=ASFLAG:PRIV).

Synchronized Write

This mode only applies to ISAM PLUS files implemented under D-ISAM. When activated, it forces any write operation to wait until the data has actually been committed to disk before returning to the program. This will dramatically slow down access but might be useful in certain networked situations where you want to eliminate the possibility of a user on another CPU reading an outdated copy of the data due to buffer synchronization problems. It may also be useful in extremely critical applications where you cannot afford the possibility that a system crash will leave a lot of uncommitted updates stranded in memory.

Allow Divide By 0

This enables the option of treating a division by zero as equal to zero, rather than generating a BASIC error. This is the runtime equivalent of the BASIC Plus compile-time source directives (DIVIDE'BY'0 and NO'DIVIDE'BY'0).

Memory Mapping

This activates automatic memory mapping for any files subsequently opened within the current program. Option 16 (Memory Map) causes the subsequently opened random files to be "memory mapped". Memory mapping is a technique used very effectively on UNIX/Linux systems to greatly speed up file access. Essentially it makes the file act like virtual memory, such that you can then access it with memory (rather than disk) operations. This eliminates most of the overhead of disk service calls (which under UNIX require a context switch to supervisor mode, and under Windows networks may require machine-to-machine messaging). The advantage is most pronounced in files which are accessed very frequently and have small record sizes. For this reason, A-Shell uses memory mapping on the qflock.sys and jobtbl.sys files under UNIX/Linux.

There are, however, two downsides to memory mapping. The biggest one only affects Windows, where the memory "view" of a file is not kept coherent with the disk "view" of the file. Thus it is not a good idea to use this technique in multi user mode, unless you are only planning to read from a file.

The second downside is that it causes a lot of memory to be allocated, which may decrease overall efficiency of the system if it is used too much. (If the total size of memory mapped files exceeds available memory, the system will begin to "thrash", which is not a pretty sight.)

No IDX Lock

This was implemented mainly as a test to see if ISAM access in a non-LOKSER environment could be significantly improved by not bothering to place a lock on the IDX "rock" when performing an ISAM operation. In theory, this simulates the way it works under AMOS when LOKSER is not on. In our experience so far, it does not seem to have that much of an effect. It was implemented as a file-oriented switch rather than a global option (like OPTIONS=ISAM_IDXLOK) because it is probably only practical, if at all, with files that are only going to be scanned and not updated. WARNING: using this option with a file that is being updated may lead to IDX corruption, unless you have some other mechanism in place to make sure two users are updating the IDX at once!

Local Copy

This only applies to Windows, and provides yet another way of speeding up access to a file that is only going to be read. When turned on, all random OPEN file operations result in a local copy being made of the file in the workstation's "TEMP" directory. (It uses the TEMP environment variable definition to locate this directory.) The local copy is then accessed, rather than the network copy. When the file is closed, the local copy is deleted (and any updates to the file are lost!)

The Local Copy option provides nearly as much performance improvement as memory mapping and even loading files into memory, and has the advantage of not requiring any extra physical memory. So it can work just as well with a 100MB file as with a 500K file (provided you have sufficient disk space). As with all of the related schemes for getting around the performance penalty of peer-to-peer shared file access, it makes the most sense in situations where you are going to be accessing a large part, or all, of the file. (It would not make much sense to transfer a 100MB file from the server to the workstation and then only access a few records from it.) Unlike memory mapping and the MEM: device, this technique works with ISAM files as well, and is immune to concerns over how well Windows supports it (since all we ask out of Windows here is the ability to copy a file).

Force Control-C to Parent

This allows **Error! Reference source not found.** subroutines to mimic a common behavior in traditional (AMOS) SBRs which do not do their own handling of Control-C. The trick is to add the following to the error trap routine of the SBX module:

```
TRAP:
   if err(0)=1 then
      xcall ASFLAG,128      ! set ^C in parent
      END                   ! exit SBX
   Endif
```

If the *flag* argument is supplied as a 2 byte binary variable, then the previous flags settings will be returned in the variable, after the new settings are updated from the variable. This technique would be important when using **ASFLAG.SBR** to change the settings for just one file, returning to the original settings thereafter.

AUI

The AUI subroutine serves as a organizing wrapper for most of the primary classes of functions of the AUI Toolkit.

```
xcall AUI, CLASS, parameters
```

We use the object-oriented "class" terminology not to claim that these meet every definition of an object class, but to promote an object orientation to the understanding and use of the functions. Adopting this mind set early on in a text-to-GUI conversion project will reinforce good design and contribute to better results.

CLASS is a string which names the class of interface "object", from the table below.

Class	Description
Control	Display and input controls. Properties include the control type (button, checkbox, etc.), size and position (using the row/col coordinate system), etc. Methods include create, modify, delete, save, restore, clear, etc. Same as XCALL MIAMEX 199.
Eventwait	Abstract class whose purpose is to provide methods for waiting for an interface event (typically a button to be clicked). Used as an alternative to INFLD or other input routine. Same as XCALL MIAMEX 135.
Environment	Properties related to the interface environment.
Menu	Menu items. Methods for adding, deleting, and enabling/disabing items. Same as XCALL MIAMEX,71
HTMLHelp	Interface with HTML/CHM help files.
Image	Special case of control for images. Same as XCALL IMAGE
Window	Main display window configuration. Properties include number of rows and columns, status lines, colors. Methods include PRINT and TAB
other	Other classes can be implemented as **Error! Reference source not found.** modules. XCALL APL,CLASS$,... will be treated internally as VXCALL CLASS$,...

Note that there are two other class-level objects which should be in this table but are not. These controls were developed before those noted in the table above, and therefore have already been documented. Rather than move that documentation, we instead refer you to those topics:

- The control for handling data input and output is INFLD.
- The control for managing trees and listboxes is XTREE.

Control

```
xcall AUI, AUI_CONTROL, opcode, ctlid, ctext, cstate, ctype, cmd,
      func, cstatus, srow, scol, erow, ecol, fgc, bgc, fontattr,
      fontscale, fontface, tooltip, parentid, winclass, winstyle,
      winstylex
```

The AUI AUI_CONTROL class provides methods for creating a wide range of Windows GUI "controls", such as buttons, static text objects, listboxes, etc.

Currently this class is supported under Windows (and ATE) only, although there is nothing prohibiting an ambitious programmer from implementing a "superclass" that provides similar capabilities in a text environment. In fact, it is generally recommended that developers create their own wrappers for these low level functions in order to standardize and accelerate their development.

Parameters

All of the parameters are optional except for the first two, *opcode* and *ctlid*, which together are sufficient to delete the control. Even when adding a new control, the last several parameters are only needed when you want to do something slightly unusual or fancy. Refer to the comments for the individual parameters below for deciding how many you need to specify for a particular effect.

Parameter	Description
opcode	[in] Determines the "method" or operation to perform.
ctlid	[in/out] Control ID (an integer starting from 1 that uniquely identifies each control).
ctext	[in/out] Text associated with the control. For most typical controls (buttons, static text, checkbox, etc.) this is the text that appears in the control.
cstate	[in] State of the control (enabled, disabled, hidden, etc.).
ctype	[in] Control type and attributes. (For more exotic controls, leave this zero and use *winclass*, *winstyle*, *winstylex*.)
cmd	[in] Specifies the command or keyboard string which is invoked when the control is clicked.
func	[in/out] When the cmd type is DLL, this can specify a function within the DLL to execute. For checkboxes, func is a 1-byte binary which gets set and cleared along with the display state of the checkbox.
cstatus	[in/out] Returns status indicating success of operation. (For some opcodes, it is used as an input also.)
srow, scol, erow, ecol	[in] These mark the upper left and lower right corners of the control in standard row/col coordinates. Note that this may not have much relationship to how many characters are displayed within the control.
fgc, bgc	[in] Foreground and background color palette numbers for the control.
fontattr	[in] Used to override the default font attributes for the control.
fontscale	[in] Scales the font to the specified percentage.
fontface	[in] (string) Overrides the default font selection.
tooltip	[in] (string) Defines a "tooltip" (popup message)
parentid	[in] If specified, new control becomes child of specified control.
winclass winstyle winstylex	[in] May be used as an alternative to ctype to create a control using the standard Windows control classes and styles.

opcode

Opcode specifies one of the "methods" or operations from the table below. The symbols are defined further in **Error! Reference source not found.**.

Symbol	Description
CTLOP_INFO	No change to the control; just check to see if exists and retrieve information about the control. This is useful, for example, if you receive the *ctlid* as a response to another function, such as AUI Eventwait, and want to identify the control by its text.
CTLOP_ADD	Add new control
CTLOP_CHG	Change state, text, and/or command of existing control
CTLOP_DEL	Delete control(s). This may be done explicitly, one at a time, by specifying the *ctlid* or implicitly, for all controls in a rectangular region, by setting *ctlid* to 0, *ctext* to "*" and the *srow, scol, erow,* and *ecol* coordinates to the boundaries of the region.
CTLOP_CLR	Clear. Similar to delete but with two important differences. One is that when used on a region (set the *ctlid* to 0 and *ctext* to "*"), clear will only delete the unprotected controls . The other is that clear may be used on a parent control to delete all of its children without deleting the parent. (This is particularly useful with TAB controls.)
CTLOP_QRYCB	Query value of checkbox. (Sets *cstatus* to 1 if checked, else 0.)
CTLOP_SVA	Save controls intersecting a specified rectangle.
CTLOP_RSA	Restore last saved set of controls
CTLOP_SBCH	Start batch. (Useful in the ATE environment to create many controls without requiring a response from the client for each control.)
CTLOP_EBCH	End batch
CTLOP_GETID	Locate control by its coordinates
CTLOP_PANE	Select a specific tab pane within a Tab control (set *cstate* to the desired tab item; 1=first).

ctlid

(integer variable, typically B,2) Returns an ID for a newly added control, and may be specified in subsequent operations as a way of identifying the control. *Ctlid* values will range from 1 to the maximum number of controls (currently 600) which might make it useful as some kind of array index within your application.

ctext

(string, up to 199 characters) Specifies the primary text that will appear on or in the control. This applies mainly to simple controls like buttons, static text controls, checkboxes, edit boxes, etc. Some controls have no text (such as a rectangle or image), or have so much text (like a TAB control or listbox), that *ctext* or is used for some other purpose, such as to specify the filename or resource name of the bitmap or icon to use in place of text. See the table of *opcodes* for specific exceptions based on the *opcode*.

As a convenience when deleting or clearing controls, you may set *ctext* to "*" to delete or clear all within the specified *srow*,*scol*,*erow*,*ecol* coordinates. Or you may specify the control purely by its *ctlid* and omit all the remaining parameters.

cstate

This indicates the initial state of the control (when adding) and flags which affect the change operation. The symbols come from **Error! Reference source not found.**.

Symbol	Description
MBST_ENABLE	Enabled
MBST_DISABLE	Disable (grayed out)
MBST_HIDE	Hidden (invisible) .
MBST_SHOW	Make visible (if hidden)
MBST_CHANGE	Change *ctext* and *cmd*. Use this with *opcode* 2 to update the *ctext* and *cmd* parameters associated with an existing button. Otherwise, only the button state is updated.
MBST_TEXTONLY	May be added to *MBST_CHANGE* to cause only the text of the control to be modified. This can eliminate the "flicker" that might otherwise appear as all of the attributes of the control are reset, especially when updating the title bar of a dialog.

In the special case of opcode 11 to set the current page of a Tab control, *cstate* should be set to an integer ranging from 1 to the number of tab items in the control.

ctype

This specifies any valid combination of options from the table below. The symbols come from **Error! Reference source not found.**.

Symbol	Description
MBF_BUTTON	(default control type) A rectangular button, which displays a text string and is virtually always associated with a click action. See Button Control.
MBF_CMDLIN	(default *cmd* type) Clicking the control causes the contents of *cmd* to be executed as a Windows command line, as if it had been executed via XCALL HOSTEX. See cmd notes below.
MBF_DLL	DLL. Clicking the button causes the function (*func*) with the associated DLL (defined in *cmd*) to be loaded and executed.
MBF_CHKBOX	Checkbox. Clicking on the button toggles the checkmark display, as well as the value of the one-byte parameter (0 for unchecked, 1 for checked) in func. See Checkbox Control.
MBF_3STATE	3 State Checkbox. Same as a regular checkbox except it has 3 states: unchecked (0), checked (1), and indeterminate (2).

Symbol	Description
MBF_AUTO RADIOBTN	Radio button. Used in a group (see MBF'GROUPBOX) of two or more radio buttons. Similar to MBF'CHECKBOX except that only one may be selected at a time. Although popular in Windows forms, from a programming standpoint, they are more complex and less efficient than and INFLD combo box (which otherwise accomplishes the same task of allowing the selection of just one from a short list of choices. See Checkbox Alignment, Justification .
MBF_LFTEXT	For controls such as checkboxes, that contain a text element and some other element, specifies that the text element appear to the left of the other element. (Default is to the right.)
MBF_LFJUST	Left justify text. (Default is center.) Note that for checkboxes, this affects only the justification of the text within the space allotted to it, not whether the text is to the left or right of the checkbox..
MBF_RTJUST	Right justify text. (Default is center.) See note for *MBF'LFJUST* above..
MBF_BITMAP	For buttons (*MBF'BUTTON*), cause *ctext* to be interpreted as the filespec (either AMOS or native) of a bitmap (BMP) to display in the button instead of text. Image will be stretched to fill the button.
MBF_SYSMENU	For dialogs (*MBF'DIALOG*), causes the dialog to sport an "X" (aka system menu) in the upper right corner allowing it to be closed by clicking on it. See Modal Dialog Box.
MBF_ICON	Icon. For buttons (*MBF'BUTTON*), cause *ctext* to be interpreted as the filespec (either AMOS or native) of an icon (ICO) file to display in the button instead of text. Image will be stretched to fill the button.
MBF_TABSTOP	Indicates that the control is part of the group of controls that may receive the focus by tabbing between them. This is automatic for MBF'BUTTON, but may optionally be applied to MBF'CHKBOX, MBF'3STATE, and MBF'AUTORADIOBTN controls. Currently this is only applicable in conjunction with the AUI EVENTWAIT class.
MBF_SUNKEN	For static text controls (*MBF'STATIC*), causes them to appear with an "sunken" edge.
MBF_KBD	Keyboard. Clicking on the control causes the contents of *cmd* to be forced into the keyboard buffer. See cmd notes below for specifications and hints on how to use this technique effectively..
MBF_SHLEXC	Shell execute. Clicking the button causes the contents of *cmd* to be interpreted as an object (file or URL) to be opened using the associated application defined in the Windows Registry. See cmd notes below.
MBF_STATIC	Static control, normally used for text prompts, but also may be used for graphic lines. See example under Add.
MBF_EDIT	Edit control (reserved for INFLD use). See Icon Control.
MBF_ALTPOS	Adjusts the way in which a control is positioned, with the specific effect varying by the type of control. For buttons (*MBF_BUTTON*) and group boxes (*MBF_GROUPBOX*) it causes a slight vertical adjustment (usually an improvement) in the way they are positioned and sized on the screen. See Groupbox + MBF_ALTPOS. For horizontal lines, it moves them from the center of the row to the bottom of the row. For dialogs, it switches the entire dialog (and all its child controls) to the Alternate Dialog Coordinate System.

Symbol	Description
MBF_WORD ELLIPSIS	For static text controls (*MBF'STATIC*), if the text in *ctext* does not fit within the control, an ellipsis (...) will be displayed.
MBF_PATH ELLIPSIS	For static text controls (*MBF'STATIC*), if the text in *ctext* does not fit within the control, it is interpreted as a path, and an ellipsis will be used for the middle portion of the path to shrink it to fit.
MBF_WRAP	Applies to buttons (*MBF'BUTTON*) only, causing text within them to wrap to multiple lines (assuming the button is tall enough). This is automatic for static controls.
MBF_NODISTORT	Applies to icon buttons (*MBF'BUTTON+MBF'ICON*) only, causing the square aspect ratio of the icon to be preserved when scaling it to the size of the button.
MBF_DIM	For static text controls (*MBF'STATIC*), applies the equivalent of the "dim" attribute. This affects the interpretation of the foreground color (*fgc*) and also makes the text behave like normal fixed-pitch text with respect to protection and clearing.
MBF_LISTBOX	Listbox control. (Reserved for internal use by XTREE)
MBF_GROUP BOX	Creates a "groupbox", which can be used to group other controls. (Do not set *MBF'STATIC*!) See Groupbox Control.
MBF_FRAME	When used with *MBF'STATIC*, creates a variation of static control that has a "frame" edge, making it stand out from the background (sort of the reverse of the sunken effect.)
MBF_COMBOBOX	Combo box control (reserved for INFLD use)
MBF_UNPRO TECTED	May be applied to buttons and other controls which are normally protected, to allow them to be cleared by opcode 4. See Clear.
MBF_PROGRESS	Progress bar control. See Progress Bar Control
MBF_DIALOG	Dialog box. See Modal Dialog Box
MBF_TAB	Tab control. See Tab Control

Alternate Dialog Coordinate System

The standard A-Shell coordinate system is based on dividing the main window size by the currently defined number of rows and columns. To put it another way, the basic unit of the coordinate grid is the text-based coordinate cell size. This fact helps facilitate the coexistence of text and GUI elements on the same window, but it has two shortcomings for GUI development:

1. The grid size is dependent on the actual size of the main A-Shell window, which can be easily adjusted, thus leading to a grid size that may be vastly too big or too small for a particular screen environment
2. It is based on fixed-pitch fonts which have little to do with the proportional fonts used in GUI environments.

The problems become particularly noticeable with dialogs, which are typically designed to be a certain size relative to the information contained within, and not dependent on the size of the window that called the dialog. Yet, with the normal A-Shell grid system, the dialog, being just another control, uses the same grid as the parent window, making the dialog layout helplessly dependent on the user setting the main window to a reasonable size.

To overcome these shortcomings, the MBF_ALTPOS option may be specified when a dialog is created in order to have the dialog use a more flexible coordinate system which is independent of the A-Shell main window and which self-adjusts to provide a "constant logical size". This is accomplished by basing the alternate grid on the current standard GUI font size. In addition to being independent of the size of the main window, it also adjusts to the "DPI" of the Windows desktop.

The units of the alternate grid are still based on individual characters, which allows you to lay out the dialog almost as if it was a traditional text screen (where one character equals one column). Only here, we are using an "average character width". So while you can count the actual number of characters in your dialog labels to determine the sizes needed for the controls, as with "standard Windows dialogs", you need to allow enough room for the variability in character widths of variable text, since capital letters are on average twice as wide as lower case in the typical GUI font. So you can't count on a 20 character string fitting in 20 "columns" of the dialog. But if you use mostly lower case, it should be a rough approximation, and if you allow a little extra space, your dialogs should be pretty resilient to local environment settings.

As an added bonus, this coordinate system is reasonably compatible with the old one, meaning that adding MBF_ALTPOS to an existing dialog willprobably not break it, and may improve it.

There is one significant difference that must be noted when using the MBF_ALTPOS option, involving the way the height of the dialog is calculated. In the original dialog implementation, the total dialog height was based on multiplying the row height of the main window by the number of requested rows in the dialog coordinates. But this calculation included the dialog border and title bar, which reduced the usable number of rows of the dialog.A margin was added to the bottom, but the net effect is that normal dialogs have one less usable row than they ought to. And when the dialog gets too small, even that last row may be truncated. Arguably this should be fixed since it is confusing, but doing so now would probably affecttoo many programs.

However, when the MBF_ALTPOS flag is specified, the dialog height is calculated to take into account the border and title bar, so that you always get the full number of rows specified. For example, if a dialog was coded to start at row 5 and end at row 15, you get 11 usable rows.(In addition, there is a half-row margin at the bottom, which gives buttons placed on the bottom row of a dialog a little breathing room.)

TRMCHR.SBR is aware of this distinction, and returns the full number of rows in WINROWS when the MBF_ALTPOS flag was used to create the dialog. Otherwise, it returns one row less, which generally works but is still an approximation since the amount of space lost due to the border and title bar could amount to more than one row if the rows are short enough.

This "correction" to the height calculation of dialogs may cause your existing dialogs to appear to have an extra row at the bottom (or, if you used the BTNMNU.SBX option to put buttons on the bottom row, then the extra space will appear above that.) The TSTEVW sample program has been upgraded to deal with this by subtracting one from the height of the dialogs if you select the MBF_ALTPOS option.

Another minor difference when using the MBF_ALTPOS option, is that the starting row/col coordinates of the dialog are interpreted as relative to the main window, and not to the parent dialog. This only matters for nested dialogs, and allows you more flexibility in positioning such nested dialogs (which otherwise can't start up or to the left of the previous dialog.) But since dialogs can be dragged around thescreen by the user, it probably won't have any significant deleterious effect on existing dialogs that are converted to the new grid system.

Note that a nested dialog will automatically inherit the ALTPOS grid size from it's parent dialog (if the parent dialog had the ALTPOS option). But it will not automatically inherit the alternate positioning logic or height

calculation. That is, if the parent dialog uses MBF_ALTPOS, then any nested child dialog will automatically use the same spacing. But unless the child dialog also uses MBF_ALTPOS, its upper left corner will be positioned relative to its parent, and its bottom row usability will be related to the relationship between the row height and title bar height. Otherwise, if the child dialog does use MBF_ALTPOS, then itsupper left corner will be relative to the main window and use the same height logic, and thus if it has the same coordinates as the parent dialog, it will exactly overlay it.

cmd

(string, up to 199 characters) specifies the command, DLL, keystrokes, or object reference (according to *ctype*) associated with clicking on the control. The choices are:

- Windows Command Line
- Keyboard Commands
- Shell execute

Windows Command Line

(*MBF_CMDLIN*) When launching a Windows command line (unrelated to A-Shell, such as the Windows calculator), it is often problematic to know the fully qualified path. If the command cannot be assumed to be in the default PATH (like NOTEPAD or CALCULATOR) or in the Registry (typical for newer installations of Office applications), then you'll probably have to use an environment variable and insist that it be defined. For example, you might put your utilities in an arbitary folder but require as part of the installation procedure that an environment variable, say, ASHUTIL, be defined, then you can specify it in your command lines (e.g. "`%ASHUTIL%\MYUTIL.EXE`"). Or you can just require that the PATH environment variable be extended to include the directory where your special utilities are found.

Another idea would be to install any such custom utilities in a directory with a known relation to the DSKn directories. Then you can refer to it using relative notation, e.g. "`..\..\ashutils\myutil.exe`". (This would assume that the "ashutils" folder as a sibling to the DSKn folders.)

If you are using the command line to launch another instance of A-Shell, as with XCALL HOSTEX, you can refer to the current A-Shell executable (and its current miame.ini file) via the macro symbol `$ASHELL`, and you can use the suffix characters available to HOSTEX. For example, `"$ASHELL -e run armenu $"` would launch a new A-Shell instance, run the program armenu, and suspend the current session until armenu exited.

Keyboard Commands

(MBF_KBD) Although you might find it convenient in menus to set your button or other control *cmd* strings to mimic what might otherwise be keypunched by the operator to make a menu selection, care must be taken to prevent the user from sending such "ordinary text" commands in the wrong context. The best way to avoid that problem is to define your keyboard *cmd* strings to send pseudo function key sequences. These will automatically be converted by INFLD (and other routines which wait for events, such as **BTNMNU.SBR** and AUI Eventwait) into exitcodes, and can be processed relatively easily by other character-level input routines.

There are two formats for defining pseudo-function key sequences:

```
chr(7) + chr(n)
```

or

```
chr(7) + chr(250) + "####."
```

The first format is limited to values of (n) from 1 (for F1) to 249. The second format is preferable, since it provides a nearly unlimited range of pseudo-function key numbers. (The #### shown can actually be any number of digits as long as it is terminated by a period.) So you can theoretically assign a different pseudo function key to every single control in an entire application, or at the very least to every control in a program. Note that in either case, if converted by INFLD or a similar routine into an exitcode, the exitcode will be the negative of the pseudo-function key number. So in the first case, the range is from –1 to –249, and in the second, from –1 to –9999999. See "Event-Driven" Procedural Programming in the *A-Shell Development Guide* for more details on how to use this form of keyboard command string to allow a program to respond to a user clicking randomly on controls.

Note that you can use the symbolic notation for control characters when defining your *cmd* string. This is particularly important in ATE, where certain control codes may have trouble passing through to the client unfiltered (especially ESC). One convenient symbolic notation is to use the caret "^" to indicate that the following character is a control character. For example, "^C" would be intepreted as Control-C or chr(3). Another method is to use one of the Virtual Key Symbolic Names (e.g. "%VK_ESC%" or "%VK_xF345%).

Shell execute

(*MBF_SHLEXC*) For launching Windows commands, this method is often preferable to the Windows Command Line method discussed above, since it eliminates the problem of determining the fully qualified path for the command. Instead, you just specify the file or object reference (or URL) and Windows launches the necessary program (or gives the user a chance to specify what program goes with that file or object type.). For example, if *cmd* = "http://www.microsabio.com", then clicking it would launch the browser and go to the specified web page. If *cmd* = "mydata.xls", clicking the button would launch whatever the user's preferred spreadsheet program was to open the spreadsheet.

If the file object specified in *cmd* does not contain a drive or directory specifier, then it will be assumed to be in A-Shell's "DOC" subdirectory (e.g. C:\VM\MIAME\DOC). Otherwise, you need to specify a native pathspec (unless the file is in the system search PATH). See MIAMEX 3: Perform FSPEC on AMOS string.

func

Func has two purposes. For type 1 (MBF_DLL), it must be a string containing the name of the function to execute within the DLL whose name was specified in cmd. For types 4 and 8 (checkboxes), it should be a B,1 variable whose value will be tied to the state of the checkbox (0=unchecked, 1=checked, 2=indeterminate). The initial value of *func* upon creating the checkbox will determine its initial check status. Thereafter, the value of the variable will change automatically and immediately as the checkbox is checked/unchecked.

Checkboxes can also be created and managed by INFLD, in which case it handles the details just described internally and interfaces with the application as if the checkbox were a Yes/No field.

cstatus

(F,6) returns a code indicating the result of the operation:

Value	Meaning
0	OK, or for *opcode* 2 indicates control was previously enabled
1	OK, or for *opcode* 2 indicates control was previously disabled
-1	Add or delete control function failed
-2	Button (*ctlid* or *ctext*) not found during change or delete operation
-3	Out of memory (unable to allocate control storage buffers)
-4	Exceeded maximum number of added controls (currently 600)
-5	No control buffer allocated
-6	Illegal *opcode*
-7	Control already exists
-8	Control update operation failed
-9	Unable to load bitmap or icon file. Note the common error of mismatching the *ctype* flag (256 or 512) with image file type (BMP or ICO)
-10	Incorrect control type for operation
-11	No batch in progress
-12	Illegal tab request or duplicate batch request
-13	Parent dialog not found

If you don't care about the returned *cstatus* or *ctlid*, you can specify the *cstatus* variable as a null string ("") (or eliminate it entirely if you don't need any of the other parameters). This prevents the routine from returning any information, which could provide a meaningful performance improvement over ATE with some UNIX servers, particularly when you are painting screens with large numbers of simple controls which were taking the place of PRINT statements (which never returned status before either).

srow, scol, erow, ecol

(numeric) These coordinates specify the position and size of the control, using traditional row/col units. If the *parentid* parameter specifies a valid parent control (such as a groupbox, tab, or dialog box), then these coordinates are taken to be relative to the upper left corner of the client area of the parent control.

Even though we aren't otherwise bound by the traditional text mode rule of one character per row/col cell, by fitting the control boundaries on to the same coordinate system, it is much easier to migrate from, or maintain compatibility with, legacy text applications.

Millirows

Beginning with A-Shell build 933 of 7 June 2005, control coordinates support "millirows". When the start row or end row coordinate to AUI_CONTROL or TAB(-10,20) commands is greater than 200, it is assumed to be in units of "millirows" (1000 to the row) rather than rows. This allows the vertical position and size of a

control to be more finely adjusted than is possible with integer rows or even with the MBF_ALTPOS option. Note that:

- This option applies only to the row coordinates, not the column coordinates (where such fine tuning is generally not necessary).
- To avoid confusion, millirows should probably not be used in conjunction with the MBF_ALTPOS on the same control. (except dialogs)
- The sample program GUILIN has examples of millirow usage.

In general you will get the same results whether you specify an *srow* value in units of rows or multiply it by 1000. There are some exceptions, though, where the default position of a row is adjusted up or down based on some internal logic, and in such cases, switching to millirows may defeat the internal logic, causing the line position to change more than expected. For example, with single row horizontal line controls, the default position is in the middle of a row. If you switch from, say, row 5 to millirow 5000, the second line will be half a row higher than the first. In general though, if you want to move a control up or down by a slight amount, multiply the *srow* value by 1000 and then offset it by some fraction of 1000. To convert *erow* to millirows, first add 1 and then multiply by 1000 (e.g. *erow* 5 is equivalent to millirow 6000).

In order for a control to occupy a single "row", its *erow* coordinate should be 1000 millirows higher than the *srow* coordinate. Some examples will make this clear:

Coordinates	Result
srow=2 : *erow*=2	Control is one row high.
srow=2000 : *erow*=3000	This is equivalent to the previous example. Note that when converting *erow* from standard rows to millirows, add 1 and then multiply by 1000. When converting *srow*, just multiply by 1000.
srow=2100 : *erow*=3100	This is equivalent to the previous example, except that it is shifted down by 1/10 of a row (100 millirows).
srow=2400 : *erow*=2400	This control is only 1 millirow high, which only makes sense in the case of lines, i.e. (SZCLASS="STATIC", DWSTYLE=SS_BLACKRECT or SS_GRAYRECT or SS_WHITERECT).

The system used here makes millirows logically analogous to pixel coordinates, except for the fact that the external leading is still subtracted from the bottom coordinate of the control. For example, consider two controls in adjacent rows. The first is specified as *srow*=3000 : *erow*=4000 and the second is *srow*=4000 : *erow* = 5000. Do the two controls touch? Normally, no. This is because we subtract the external leading from the resulting bottom coordinate. So if the external leading was 6, the two controls would still be separated vertically by 6 pixels, even though the bottom coordinate of the top control matches the top coordinate of the bottom control. This is a bit strange perhaps, but is very handy if you purpose in using millirows is simply to shift an object up or down by a small amount.

To cancel the automatic adjustment of the bottom coordinate to account for the external leading, specify the MBF_ALTPOS flag in the *ctype* parameter. When used in conjunction with an *erow* specified in millirows, the usual interpretation (i.e., causing an "aesthetic adjustment" to the size or position of certain control types) is changed to simply disable the automatic subtracting of the leading from the bottom coordinate.

The 1.0(103) update of the GUILIN sample program in the SOSLIB illustrates this difference by drawing a pair of columns made up of lines 1000 millirows high (in the lower right corner of the screen). The first

column has gaps between the lines due to the leading; the second column uses MBF_ALTPOS to eliminate them. Note that we could also have eliminated them by just adding more to the *erow* coordinate; this is less convenient though since we don't have a good way of knowing the leading or converting it from pixels to millirows. (Overlapping wouldn't matter for lines but it might for some other controls.)

Notes

- Millirow 1000 is the top of the usable area of a dialog. It might have been more logical for millirow 0 to have been the top, but this offset does provide some advantages. One is that it seems more consistent with the fact that the first row in standard coordinates is 1, not 0. This makes it somewhat easier to mentally convert from ROW x to the equivalent millirow (1000x), and dovetails nicely with the fact that static text is currently aligned at the top of the control area. Consequently, if you want to shift a text label up by 1/4 row, just multiply the *srow* by 1000 and subtract 250. Another advantage is that it eliminates the ambiguity that would otherwise arise if trying to position a control within 200 millirows of the top of a dialog. (If you needed to use a millirow value of between 1 and 200 to do that, it would have been otherwise intepreted as a standard row.)
- In the main window (i.e. not a dialog), there will be left, top, right and bottom margins which are made up of the pixels left over after determining the largest integer grid size that will accommodate the rows and columns required. Thus, millirow 1000, just like row 1, will not necessarily be the very top of the window. You can specify a millirow value of between 200 and 1000 to shift the control up into that margin space, although it may get clipped if your adjustment is more than the margin.

fgc, bgc

(numeric) These parameters allow you to override the default control foreground (text) and background colors, in some cases. (Some controls don't allow override colors, and most controls don't when XP Themes are active.) Specify these values as –1 to use the "current" colors (if applicable to the control type), or –2 to use the default Windows colors. Otherwise specify palette colors in the range of 0-15 (same as would be used with the TAB(-2,x) and TAB(-3,x) commands).

Static text controls are the only type whose foreground color can be overridden in all cases. However, if you want to override the color even with XP Themes are active, you must add 64 to the desired palette #. For example, setting the *fgc* value to 3 would normally give you magenta when XP Themes are not active, or the Theme default color when Themes are active. If you set the *fgc* value to 3+64, you will get magenta in all cases. Changing the background color isn't recommended, as it usually results in an "unnatural" look.

fontattr

(numeric) May be used to override the default font attributes for the control. Again, as with colors, this is probably used most effectively only with static text controls. Choose one or more (sum) from the following table. Note that these are only requests to the font mapper – if it can't supply an exact match for the given *fontface*, *fontscale*, and *fontattr* values, if will supply the "closest" match.

Value	Meaning
0	Normal (upright)
1	Italic
2	Underline

Value	Meaning
4	Strikeout
64	Extremely light
128	Extra light
192	Light
256	Normal (default if no other values between 64 and 576)
324	Medium
384	Semi-bold
448	Bold
512	Extra bold
576	Heavy
131072	Use symbol character set (instead of the ansi character set). This normally would only make sense with a fontface like “Wingdings” or “Symbol”. Symbol fonts are mainly used in the context of AUI as a trick for creating buttons with graphic, without having to find or create an icon or bitmap.

fontscale

(numeric) If specified, causes the font for the control to be scaled by the specified percentage (100=100%). This factor is applied in addition to the global font scale factor found in the Misc. Settings menu, so should only be used to adjust the font size of a particular control relative to the others.

Note that what is being scaled in this case is the point size of the font, and that may not correspond linearly to what you might expect. For example, if you want to create a static text control with double-high characters, and have specified coordinates with *erow* = *srow* + 1, you may find that a *fontscale* factor of 200 is bigger than you want. (Or it might fit the height but be too wide.) Some experimentation is in order here, and it is also highly recommended to be conservative (i.e. go for something smaller than the maximum size that will fit), because different resolution screens and other display parameters outside of your application’s control may affect the sizing. (It’s better for a message to be slightly smaller than you wanted, than for it to be so big that it doesn’t fit, or wraps inappropriately.)

fontface

(string) If specified, it is added to the request to the font mapper to help determine the font to be used. If left blank, the default is the font that is specified as the "GUI (Control)" font in the A-Shell Settings menu. If that is also blank (as it usually is), then the default becomes the standard Windows dialogs (aka "MS Dialog"). If a font face is specified but there is no such font defined to the system, the Windows font mapper will use its own reasoning to decide what font to use. To avoid the uncertainties that might entail, it is best to stick with fonts that are nearly universal, such as "Arial", "Times New Roman", etc. Note that a particular font face is generally either fixed pitch (e.g. "Andale Mono", "Courier New", "Lucida Console") or proportional ("MS Dialog", "Arial", etc.)

tooltip

(string) If specified, and if the control has a clickable, then the specified string will appear automatically just above or below the control when the mouse hovers over it for about 1 second. This is useful for additional explanation about what will happen when the control is clicked.

If you want to associate a tooltip with a static text control, it must have the MBF_KBD style flag and a click string defined in the Alternate Dialog Coordinate System parameter. (Otherwise the control isn't really "clickable", or more precisely, the click action is ignored and so is the tooltip feature.) If you want to have a tooltip even though there is no click action, the workaround is to use "VK_NULL" from the Virtual Key Symbolic Names table.

parentid

(integer) If specified when creating a new control, the new control becomes a child of the specified control. This has two primary ramifications: 1) the coordinates of the child control are taken to be relative to the client area of the parent, and 2) such child controls will be automatically deleted when the parent is deleted or cleared. (If a dialog box is currently displayed, then the *parentid* of any new controls is automatically set to the current dialog box.)

winclass

(string) May be used, along with *winstyle* and *winstylex* as an altermate way of specifying a control type. Note that this technique is only practical for controls that don't require special processing by A-Shell. (For example, it can be used safely to create a new style of button or static text control, since these do not require any special handling other than to process their click command. But it cannot be used to create a complex control like a spreadsheet, up-down, or property sheet, since in order to be used effectively, these would require integration with other A-Shell operations.) See Rectangles and Lines.

winstyle

(integer) May be used in conjunction with winclass or ctype to specify window-style flags. This is equivalent to the "dwStyle" parameter in the WIN32 API CreateWindowsEx() function (consult the Microsoft documentation for details). As with winclass, these flags are only practical if they affect the appearance or other characteristic of the control without changing the way it interacts with the application.

winstylex

(integer) Another set of flags of the same type as winstyle. These correspond to the "dwStyleEx" parameter in the WIN32 API CreateWindowsEx() function.

For what it's worth, the addition of this second set of flags is what distinguishes CreateWindowEx() from the earlier CreateWindow(); even Microsoft is guilty of not seeing far enough in the future to prevent having to add more parameters to functions over time. We have the advantage, though, of being able to use variable length argument lists, so at least we don't have to create a whole new "Ex" version of a function just to add another parameter.

Control Methods (Opcodes)

Opcode	Description
0	Query
1	Add
2	Change
3	Delete
4	Clear
5	Query Checkbox
6/7	Save/Restore Controls
8/9	Batch Operations
10	Get Control ID
11	Set Current Tab Pane

Query

Use opcode 0 to query the details of a control by its *ctlid*. This is mainly useful in three kinds of situations:

- You need to determine if a control (with a specific *ctlid*) still exists (perhaps upon return from a subroutine that may or may not have cleared the screen).
- You have a *ctlid* (perhaps returned from the BTNMNU.SBX or AUI Eventwait) but your program doesn't keep track of controls by their *ctlids*, so you need to find out more information about the control to decide what to do.
- You have a *ctlid* and you know what control it is, but the control has changeable text and you want to determine what the current text associated with the control is. This would mainly apply to a designed to be modified by the user (such as an edit control), and in that case, you would probably use INFLD and keep track of the text a different way, but this capability may be useful in an advanced situation.

To query a control, you need to specify the *ctlid*. The contents of all the other parameters are ignored on input, and are returned with updated values based on the control specified. If the control exists, *cstatus* will be set >= 0. If the control is not found, it will set to –2. Other negative values correspond to errors (see cstatus above for a table of error codes). You must specify at least the parameters up to *cstatus* (and it must be a real variable); all others are optional (but will be updated if passed).

Add

To add or create a control, use opcode 1 and specify at least all the parameters up through the *ecol* parameter. The rest are optional.

A simple example would be creating a button occupying the rectangle from (5,10) to (6,20), displaying "OK", and transmitting the equivalent of the F2 key when clicked:

```
xcall AUI, AUI_CONTROL, 1, ctlid, "ok", mbst_enable, mbf_button + mbf_kbd,
   "%vk_f2%", "", cstatus, 5, 10, 6, 20
```

The *ctlid* is ignored on input, and set on output, along with *cstatus*, to indicate the success of the operation. If you aren't interested in the returned *ctlid* or *cstatus*, you can specify a literal 0 for *ctype* and a string (rather than numeric variable) for *cstatus*. (This may be appropriate where there is little doubt that the operation will succeed, and you aren't storing the ctlid anyway. The main advantage of this technique is in UNIX environments, where it eliminates the need for the server to wait for a response from the ATE client, thus speeding up the operation of creating controls, particularly over slow connections. (See opcode 8 below for another optimization technique.) An example is given below.

Normally, the control type is determined by the *ctype* parameter, but in certain circumstances you can set *ctype* to 0 and specify the control type in the *winclass, winstyle,* and *winstylex* parameters. This technique is generally only useful for control types that do not require any interactivity (besides a click action), with the main example being static lines/rectangles/frames. For example, the GUILIN sample program (available for download from the A-Shell Shared Open Source Library), uses the following code to create all of the various lines, rectangles and frames in the screen shot which follows:

```
MAP1 DWSTYLE,B,4
MAP1 NOSTATUS$,S,1,""        ! dummy to eliminate need for returned status
. . .
CREATE'STATIC:
   xcall AUI, AUI_CONTROL, 1, 0, "", 0, 0, "", "", NOSTATUS$, SROW, SCOL, EROW,
      ECOL, -2, -2, 0, 0, "", "", 0, "STATIC", DWSTYLE, 0
```

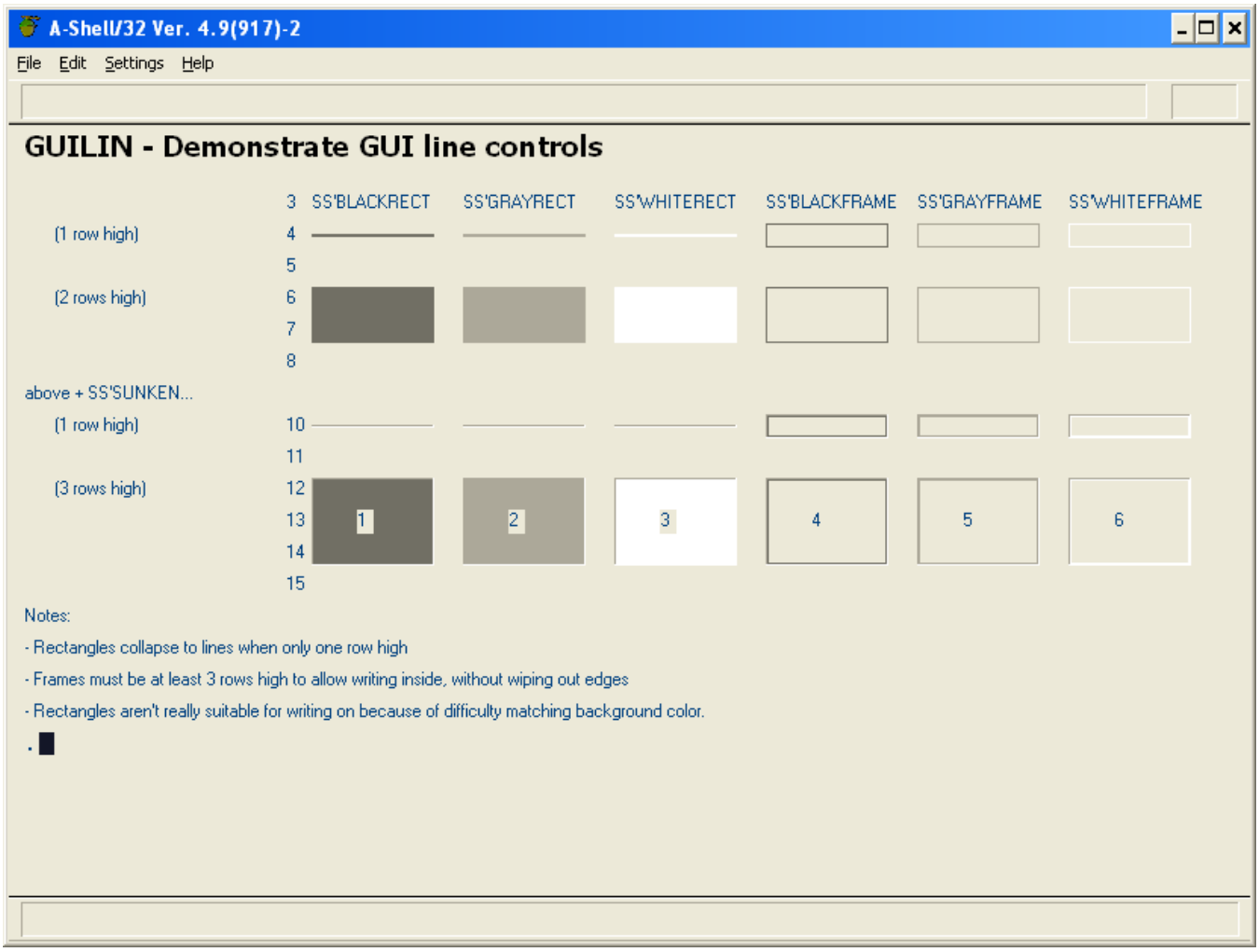
In this case, since we have no doubt about the operation succeeding, and don't care about the returned *ctlids*, we set the *ctlid* parameter to a literal 0 and use NOSTATUS$ for the *cstatus* parameter.

The different styles of lines and boxes are accomplished by setting DWSTYLE to one of the following values (defined in **Error! Reference source not found.**):

```
map2 SS'BLACKRECT,b,1,4
map2 SS'GRAYRECT,b,1,5
map2 SS'WHITERECT,b,1,6
map2 SS'BLACKFRAME,b,1,7
map2 SS'GRAYFRAME,b,1,8
map2 SS'WHITEFRAME,b,1,9
```

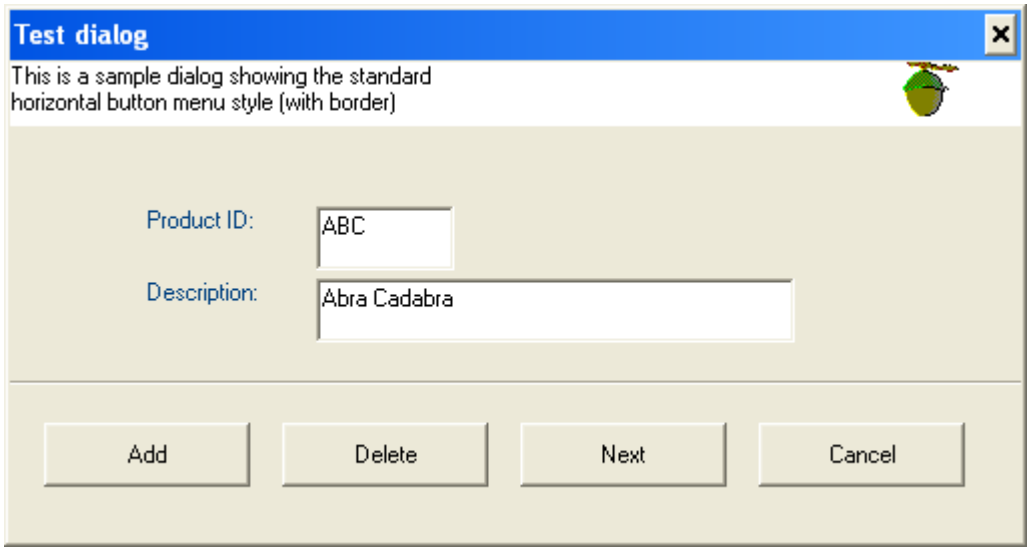
The second set (with the "sunken" look) is accomplished by adding SS'SUNKEN to one of the above:

```
map2 SS'SUNKEN,b,2,4096
```



Change

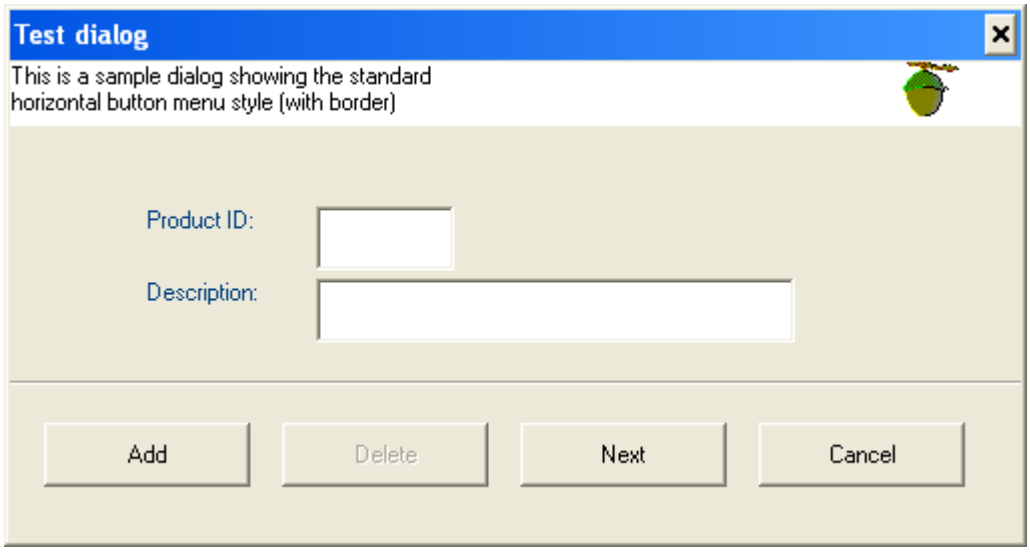
Opcode 2 can be used to change the text of a control, the command associated with it, or the state (enable/disable). For example, consider the following sample dialog:



If the data fields were blank, the "Delete" button should probably be disabled (since there would be nothing to delete). To do that, we can use opcode 2, as shown below. Note that for the purposes of this example, we will assume that the variable BTNID(2) had been previously set to the *ctlid* for the "Delete" button.

```
xcall AUI, AUI_CONTROL, 2, BTNID(2), "", MBST_DISABLE
```

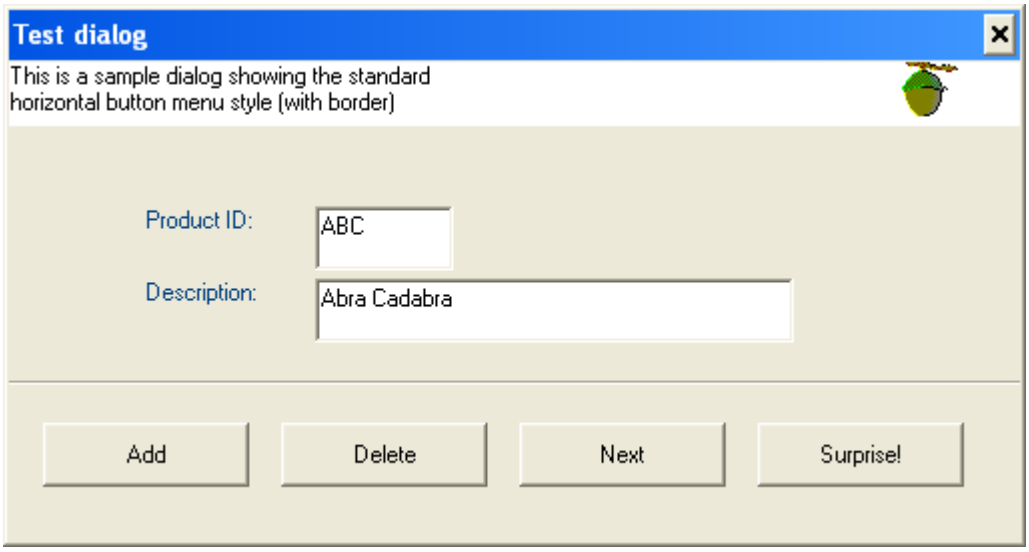
In the example above, we used the *MBST_DISABLE* value in the *cstate* parameter to change the state of the button to disabled. With this flag, no other changes are made to the control, so we don't need to worry about inadvertently changing the *ctext* and *cmd* parameters. The resulting dialog would look like this:



However, if we wanted to change the text of one of the controls (for example, changing the "Cancel" button to say "Surprise!" instead), we would need to use the *MBST_CHANGE* flag and set *ctext* accordingly:

```
xcall AUI, AUI_CONTROL, 2, BTNID(4), "Surprise!", MBST_ENABLE + MBST_CHANGE
```

The result is:



Also see Get Control ID for another example scenario in which an EDIT (INFLD) control is enabled or disabled based on the state of a radio button.

Delete

Opcode 3 is used to delete a control, either by its ctlid or by its location. Using the example above, we could delete the "Delete" button (instead of disabling, or even hiding it) as follows:

```
xcall AUI, AUI_CONTROL, 3, BTNID(2)
```

Note that unless you want to check the returned cstatus to confirm that there wasn't an error, you only need the above parameters.

As a special convenience, you can delete the current dialog (and everything in it), without knowing its ID, by setting *ctlid* = 0 and specifying *MBF_DIALOG* in the *ctype* field:

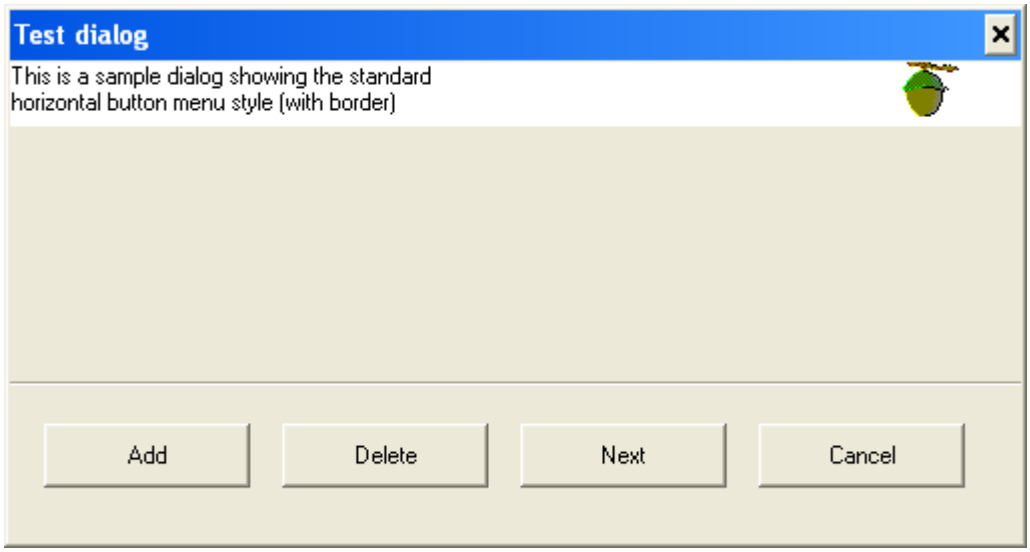
```
xcall AUI, AUI_CONTROL, 3, 0, "", 0, mbf_dialog
```

> Deleting a control will automatically delete any child controls of that control. If you want to delete just the children but leave the target control along, see opcode 4.

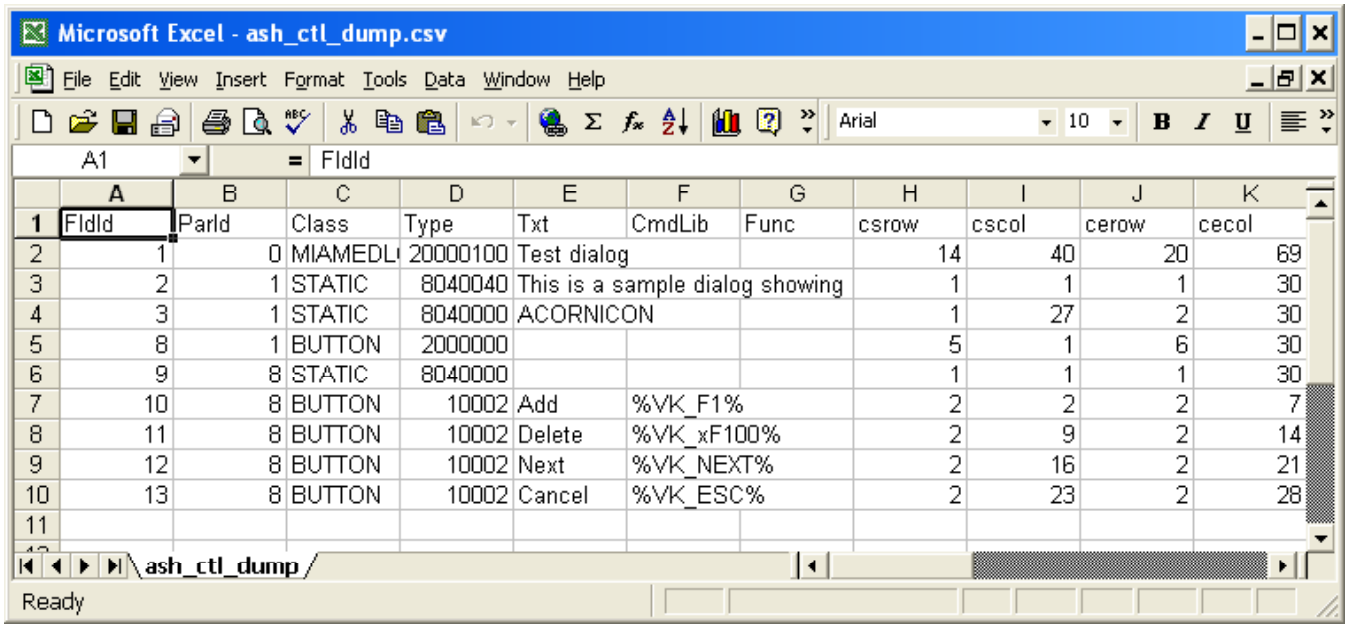
The other way to delete controls is by setting *ctext* to "*" and using the *srow, scol, erow,* and *ecol* parameters to specify a region. We can use this technique, for example, to delete the four controls in the middle of our sample dialog (Product ID, Description, and the two data fields). However, in this case, we would also have to specify the control ID of the dialog (in the *parentid* parameter), which we'll assume was stored in DLGID. (Without this, the delete operation would only look at controls that were placed directly on the main window.)

```
srow = 3 : scol = 1 : erow = 4 : ecol = 40  ! coordinates of area to delete

xcall AUI, AUI_CONTROL, 3, 0, "*", cstate, ctype, cmd, func, cstatus, srow, scol,
   erow, ecol, fgc, bgc, fontattr, fontscale, fontface, tooltip, DLGID
```

(The parameters shown above in lower case are ignored for this call; we only specified them because we need to have valid placeholders for the parameters up to the parentid parameter (specified as DLGID above). We could have used zeroes and "", but this makes it more clear.) The result is:



Note that when deleting controls by their coordinates, if the controls are part of a dialog or other group, then you need to specify the parent control ID in the *parentid* parameter, and use coordinates relative to the parent control. In the example above, we used coordinates relative to the dialog, which was the parent of the controls we wanted to delete. If, for example, we wanted to delete the buttons from the dialog, we would need to know whether they were part of a group. Often, buttons arranged as shown above are part of a group that may no be obvious visually, but can be determined by looking at the code or by dumping the controls by using Control+Shift+Double-RightClick somewhere on the dialog or main window. For the example above, the resulting spreadsheet looks like this:



	A	B	C	D	E	F	G	H	I	J	K
1	FldId	ParId	Class	Type	Txt	CmdLib	Func	csrow	cscol	cerow	cecol
2	1	0	MIAMEDL	20000100	Test dialog			14	40	20	69
3	2	1	STATIC	8040040	This is a sample dialog showing			1	1	1	30
4	3	1	STATIC	8040000	ACORNICON			1	27	2	30
5	8	1	BUTTON	2000000				5	1	6	30
6	9	8	STATIC	8040000				1	1	1	30
7	10	8	BUTTON	10002	Add	%VK_F1%		2	2	2	7
8	11	8	BUTTON	10002	Delete	%VK_xF100%		2	9	2	14
9	12	8	BUTTON	10002	Next	%VK_NEXT%		2	16	2	21
10	13	8	BUTTON	10002	Cancel	%VK_ESC%		2	23	2	28
11											

The first column lists the control IDs, and the second column indicates the parent of the control. In this case, all the controls except the dialog itself have a parent. But the buttons are actually children of control #8,

which shows as "BUTTON" in the spreadsheet but is actually an invisible *MBF_GROUPBOX*. (For reasons only known to Microsoft, group boxes share the BUTTON class with more traditional buttons and check boxes.) So if we had wanted to delete the buttons using their coordinates, we would have need to specify the correct *parentid* (8) and the correct coordinates relative to the parent (i.e. they are all on row 2).

An easy way to delete all controls is by clearing the screen with TAB(-1,0). This is equivalent to specifying opcode 3 with *ctext* = "*" and the coordinates set to the entire screen.

Also note that the coordinates do not have to be precise; they just have to specify an area that includes the controls to be deleted. You can set the *erow* and *ecol* parameters to 999,999 if you want to clear to the end of the screen or dialog without know its size.

Clear

Opcode 4 is just like opcode 3, except for the following differences:

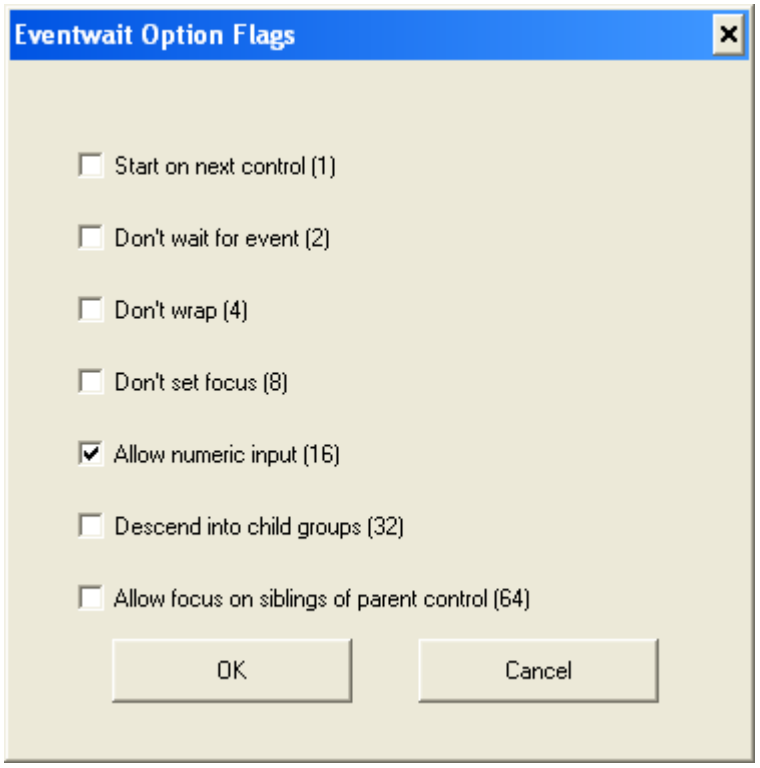
- If *ctlid* is specified, then only the children of that control are deleted. This is most useful with TAB controls, where, as you change panes, you need to remove the children previously displayed on the old pane and create the ones corresponding to the new one. If *ctlid* does not refer to a control with children, then it is ignored (i.e. treated as if zero), in which case it identifies the controls to be deleted by the coordinates (assuming *ctext* = "*").
- It only deletes "unprotected" controls. By default, static text controls are unprotected, while most other types are protected. (You can, however, remove protection from protected controls by adding *MBF_UNPROTECTED* when the control is created.) To add protection to text controls, you have to add the dim attribute and also enable protection just like you would with regular text, using TAB(-1,13). For text controls created with AUI, AUI_CONTROL, you add protection by specifying *MBF_DIM* when creating the control. For text controls created via TPRINT commands, the MBF_DIM attribute is set automatically if the current text mode is dim (i.e if TAB(-1,11) was issued previously.) The point of all this is to allow you to perform the same "trick" that is possible with dumb terminals, whereby you can protect the background of a form and clear the foreground with just a couple of commands. If it just seems confusing, you can probably ignore it.

PRINT TAB(-1,10) (clear to end of screen) and PRINT TAB(-1,9) internally perform an opcode 4 for the region in question.

Query Checkbox

Opcode 5 is similar to opcode 0, except that it is only applicable to checkboxes and radio buttons, and only returns an indication of whether the button is checked or not by setting *cstatus* to 0 for unchecked, 1 for checked, or 2 for indeterminate. (The indeterminate state only applies to MBF_3STATE checkboxes.) As with other opcodes, a negative cstatus indicates an error.

The ability to query the value of a checkbox is useful when you want to present a bunch of checkboxes and let the user click on them in any order, without having to code all the INFLD exitcode logic necessary to support arbitrary cursor motion. For example, consider the following simple dialog (from the TSTEVW sample program in the source library):



The dialog above could be created with code similar to the following:

```
MAP1 CID(7),B,2       ! control Ids for the checkboxes
MAP1 CB(7),B,1        ! chkbox state (0=unchecked, 1=checked)
MAP1 OPTDLGID,B,2     ! ID of dialog box
MAP1 STATUS,F,6
MAP1 NOSTATUS$,S,1,""  ! use when we don't care about status or ID

CREATE'DIALOG:
   OPTDLGID = 0
   xcall AUI,AUI_CONTROL, 1, OPTDLGID, "Eventwait Option Flags", MBST_ENABLE,
      MBF_DIALOG + MBF_SYSMENU, "", "", STATUS, 2, 2, 11, 23

if OPTDLGID = 0 then goto ERROR'OUT

!(assume that CB(1-7) values have been initialized appropriately)

xcall AUI, AUI_CONTROL, 1, CID(1), "Start on next control (1)", MBST_ENABLE,
   MBF_CHKBOX + MBF_LFJUST, "", CB(1), STATUS, 2, 3, 2, 21, -2, -2, 0, 0, "", "",
   OPTDLGID

xcall AUI, AUI_CONTROL, 1, CID(2), "Don't wait for event (2)", MBST_ENABLE,
   MBF_CHKBOX + MBF_LFJUST, "", CB(2), STATUS, 3, 3, 3, 21, -2, -2, 0, 0, "", "",
   OPTDLGID

xcall AUI, AUI_CONTROL, 1, CID(3), "Don't wrap (4)", MBST_ENABLE, MBF_CHKBOX +
   MBF_LFJUST, "", CB(3), STATUS, 4, 3, 4, 21, -2, -2, 0, 0, "", "", OPTDLGID

xcall AUI, AUI_CONTROL, 1, CID(4), "Don't set focus (8)", MBST_ENABLE, MBF_CHKBOX +
   MBF_LFJUST, "", CB(4), STATUS, 5, 3, 5, 21, -2, -2, 0, 0, "", "", OPTDLGID
```

```
xcall AUI, AUI_CONTROL, 1, CID(5), "Allow numeric input (16)", MBST_ENABLE,
   MBF_CHKBOX + MBF_LFJUST, "", CB(5), STATUS, 6, 3, 6, 21, -2, -2, 0, 0, "", "",
   OPTDLGID

xcall AUI, AUI_CONTROL, 1, CID(6), "Descend into child groups (32)", MBST_ENABLE,
   MBF_CHKBOX + MBF_LFJUST, "", CB(6), STATUS, 7, 3, 7, 21, -2, -2, 0, 0, "", "",
   OPTDLGID

xcall AUI, AUI_CONTROL, 1, CID(7), "Allow focus on siblings of parent control
   (64)", MBST_ENABLE, MBF_CHKBOX + MBF_LFJUST, "", CB(7), STATUS, 8, 3, 8, 21, -
   2, -2, 0, 0, "", "", OPTDLGID

xcall AUI, AUI_CONTROL, 1, 0, "OK", MBST_ENABLE, MBF_BUTTON + MBF_KBD, "%VK_F2%",
   "", NOSTATUS$, 9, 4, 9, 10, -2, -2, 0, 0, "", "", OPTDLGID

xcall AUI, AUI_CONTROL, 1, 0, "Cancel", MBST_ENABLE, MBF_BUTTON + MBF_KBD,
   "%VK_ESC%", "", NOSTATUS$, 9, 13, 9, 19, -2, -2, 0, 0, "", "", OPTDLGID
```

Having created the dialog, we can allow the user to click on the buttons without any further program intervention.

When the user clicks on the OK button (which we can detect by the fact that it was defined to act like the F2 key), we can query the values of the checkboxes with the following code:

```
For I = 1 to 7
   xcall AUI, AUI_CONTROL, 5, CID(I), "", 0, "", "", STATUS
   If STATUS >= 0 then
      CB(I) = STATUS     ! 0=unchecked; 1=checked;
   Else
      ? "Error querying checkbox"
   Endif
Next I
```

Under A-Shell/Windows, the above operation to query the checkboxes would not actually be necessary, since it will set the CB(x) variables directly in real time as the checkboxes are checked. However, that feature isn't supported when the application is running on a server other than the local workstation, so the query method is more universal.

Also see Get Control ID for an example scenario involving the need to query a radio button in order to enable/disable another control.

Save/Restore Controls

Opcodes 6 and 7 are normally only used internally by A-Shell, in conjunction with command which save and restore full or partial screen areas (e.g. TCRTs 148, 149, 202, 203, plus pop-up subroutines like MSBOXX, INMEMO, etc.) If for some reason you wanted to do this directly, note the following:

- You must specify up through the *ecol* parameter.
- The func parameter should be set to the string representation of a numeric ID (e.g. "1"). The same ID must be specified on the restore operation as for the save (this is how the correct set of saved controls is located.)

Batch Operations

Batch operations offer increased efficiency for creating many controls at a time, and may be of interest when the application server and workstation are not the same machine. In that environment, for each control created, the application server sends the request to the workstation, which creates the control and sends back the control ID and status. Although the amount of information transferred is minimal, the delay introduced by the server having to wait for a response to each command before issuing the next can add up to noticeable delays, particularly on a slow network.

There are two general techniques for minimizing those delays: activating the "ATE Reverse Channel" (described elsewhere), and creating controls in a batch. When controls are created in a batch, the return information (ID and status) for each control is buffered on the workstation until the batch is complete, after which it sends all the return information in a single packet. The main drawback of this approach is simply that it takes two extra steps for each batch – one command to start the batch and another to finish it.

To start a batch, use opcode 8, as follows:

```
xcall AUI, AUI_CONTROL, 8, cstatus
```

The normal return *cstatus* is 0. If a batch has already been started, *cstatus* will be set to –12. Other values all correspond to errors.

Once the batch is started, you can proceed to issue commands to create or query controls as you normally do, but don't bother checking for returned *ctlid* or *cstatus* values, as they are buffered by the ATE client. (You do need to specify valid ctlid and cstatus parameters, just as if you were getting those parameters back on every call; otherwise the return status is not added to the buffer for later return.)

To complete the batch and retrieve the buffered values, use opcode 9:

```
COUNT = <max # of controls in batch>
XCALL AUI, AUI_CONTROL, 9, CSTATUS, COUNT, ID(1), CB(1)
If CSTATUS # <number of controls requested> then goto Error
```

The parameters should be mapped something like the following:

```
MAP1 ID(20),B,2    ! array of control ID's (sized large enough for the batch)
MAP1 CB(20),B,1    ! array of status codes (or checkbox values)
MAP1 CSTATUS,F,6   ! returns # of controls actually created in batch
MAP1 COUNT,B,2     ! number of controls expected (max)
```

The *COUNT* parameter must be set to the maximum number of controls in the batch (and no larger than the arrays.) Typically you would set this to the actual number of controls you attempted to create in the batch, although it might also be set to the maximum size of the arrays. (It just sets an upper limit on the amount of information returned.)

CSTATUS will be updated to the actual count of controls created in the batch. (If it doesn't agree with the number you tried to create, then there was a problem.)

The *ID()* array receives the *ctlid* values of the controls created in the batch.

The *CB()* array receives the status codes that would have otherwise been returned in the *cstatus* parameter for each control creation operation. Note that it must be mapped as an array of B,1 (as shown above), so negative cstatus values will appear as 255 (-1), 254 (-2), etc. (Mainly you only care if the status is 0 or not.)

Note that you normally specify the first element of each array in the parameters passed to the xcall (as shown in the example above). But if you have one large array of control Ids, yet for some reason you use multiple batches to create your controls, you could specify a different starting value for a subsequent batch, e.g. *CB(10)* instead of *CB(1)*, as long as the array is big enough.

As an example of using a batch, we will recode the example given for opcode 5 above, this time using a batch operation to create the checkboxes, and another to query them. (See the sample program TSTBMN for a complete working version of this logic).

```
MAP1 CID(7),B,2     ! control Ids for the checkboxes
MAP1 CB(7),B,1      ! chkbox state (0=unchecked, 1=checked)
MAP1 OPTDLGID,B,2   ! ID of dialog box
MAP1 STATUS,F,6
MAP1 NOSTATUS$,S,1,""  ! use when we don't care about status or ID

CREATE'DIALOG:
   OPTDLGID = 0
   xcall AUI,AUI_CONTROL,1,OPTDLGID, "Eventwait Option Flags", MBST_ENABLE,
      MBF_DIALOG + MBF_SYSMENU, "", "", STATUS, 2, 2, 11, 23
   if OPTDLGID = 0 then goto ERROR'OUT

   !(assume that CB(7) values have been initialized appropriately)

   ! Start a batch:
   xcall AUI, AUI_CONTROL, 8, STATUS
   if STATUS # 0 goto ERROR'OUT

   xcall AUI, AUI_CONTROL, 1, CID(1), "Start on next control (1)", MBST_ENABLE,
      MBF_CHKBOX + MBF_LFJUST, "", CB(1), STATUS, 2, 3, 2, 21, -2, -2, 0, 0, "",
      "", OPTDLGID

   xcall AUI, AUI_CONTROL, 1, CID(2), "Don't wait for event (2)", MBST_ENABLE,
      MBF_CHKBOX + MBF_LFJUST, "", CB(2), STATUS, 3, 3, 3, 21, -2, -2, 0, 0, "",
      "", OPTDLGID

   xcall AUI, AUI_CONTROL, 1, CID(3), "Don't wrap (4)", MBST_ENABLE, MBF_CHKBOX +
      MBF_LFJUST, "", CB(3), STATUS, 4, 3, 4, 21, -2, -2, 0, 0, "", "", OPTDLGID

   xcall AUI, AUI_CONTROL, 1, CID(4), "Don't set focus (8)", MBST_ENABLE,
      MBF_CHKBOX + MBF_LFJUST, "", CB(4), STATUS, 5, 3, 5, 21, -2, -2, 0, 0, "",
      "", OPTDLGID

   xcall AUI, AUI_CONTROL, 1, CID(5), "Allow numeric input (16)", MBST_ENABLE,
      MBF_CHKBOX + MBF_LFJUST, "", CB(5), STATUS, 6, 3, 6, 21, -2, -2, 0, 0, "",
      "", OPTDLGID

   xcall AUI, AUI_CONTROL, 1, CID(6), "Descend into child groups (32)",
      BST_ENABLE, MBF_CHKBOX + MBF_LFJUST, ", CB(6), STATUS, 7, 3, 7, 21, -2, -2,
      0, 0, "", "", OPTDLGID

   xcall AUI, AUI_CONTROL, 1, CID(7), "Allow focus on siblings of parent control
      (64)", BST_ENABLE, MBF_CHKBOX + MBF_LFJUST, ", CB(7), STATUS, 8, 3, 8, 21, -
      2, -2, 0, 0, "", "", OPTDLGID

   ! finish the batch

   COUNT = 7   ! # of controls in the batch
   Xcall AUI,AUI_CONTROL,9,STATUS,COUNT,CID(1),CB(1)
   If STATUS # 7 then goto ERROR'OUT

   ! Add the buttons (they could have been part of the batch, but
```

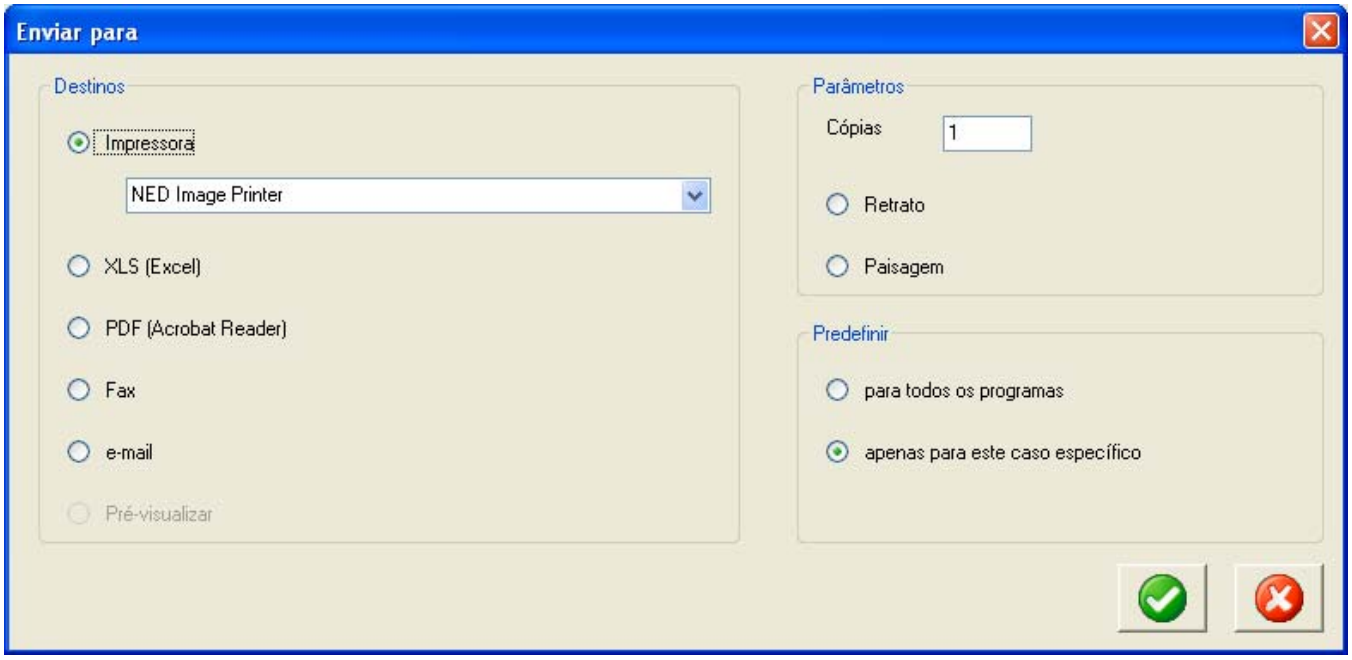
```
! here we decided to leave them out because we don't need to
! reference then directly later anyway)

xcall AUI, AUI_CONTROL, 1, 0, "OK", BST_ENABLE, MBF_BUTTON + MBF_KBD, %VK_F2%",
    "", NOSTATUS$, 9, 4, 9, 10, -2, -2, 0, 0, "", "", OPTDLGID

xcall AUI, AUI_CONTROL, 1, 0, "Cancel", BST_ENABLE, MBF_BUTTON + MBF_KBD,
    %VK_ESC%", "", NOSTATUS$, 9, 13, 9, 19, -2, -2, 0, 0, "", "", OPTDLGID
```

To query the checkboxes later, we can also use a batch operation:

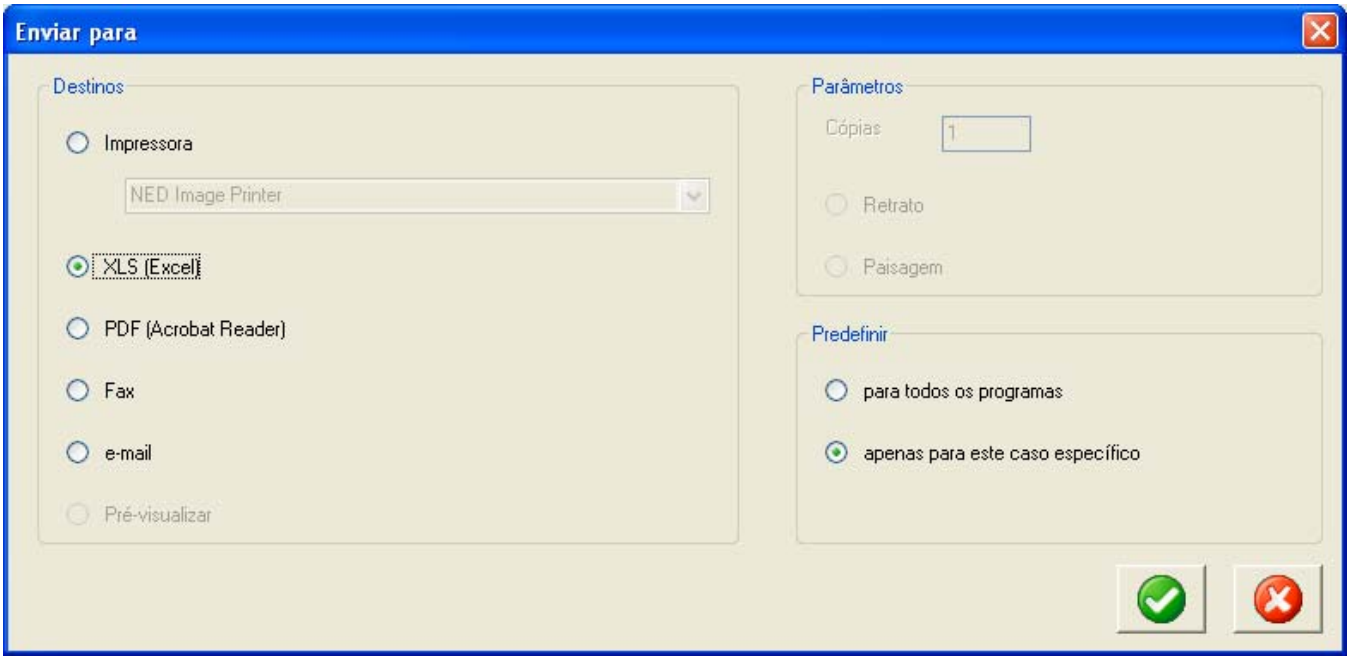
```
Xcall AUI,AUI_CONTROL,8,STATUS   ! start a batch
If STATUS # 0 goto ERROR'OUT

COUNT = 7        ! send 7 query commands...
For I = 1 to COUNT
   Xcall AUI, AUI_CONTROL, 5, CID(I), "", 0, "", "", STATUS
Next I

Xcall AUI,AUI_CONTROL,9,STATUS,COUNT,CID(1),CB(1)   ! finish batch
If STATUS # COUNT then goto ERROR'OUT

! CB(I) array is now set with the status of each checkbox
```

Get Control ID

Use opcode 10 to retrieve the ID of a control by its coordinates. This is useful when you know a control's coordinates, but not it's ID, and you need the ID to perform some operation on it (like enable/disable). For example, you might have a form which includes INFLD fields depending on certain radio buttons, as in the example below. Here is the initial form, with the INFLD controls enabled:



And here, after we change the radio button to the XLS option, we disable the controls associated with the printer:



The code sample below illustrates the creation of the "Cópias" edit control (using INFLD) and obtaining its ID:

```
COPIES:
    XCALL INFLD,2,8,XMAX,XMIN,TYPE$,ENTRY,INXCTL,1,GRPID2+1,OP,EXITCODE
    ! now get ID by its coordinates
    XCALL AUI,AUI_CONTROL,10,0,"",0,0,"","",CSTATUS,2,8,2,8+XMAX-1
    IF CSTATUS > 0 THEN
        COPIES'ID = CSTATUS
    ELSE
        PRINT "UNABLE TO OBTAIN ID OF COPIES EDIT CONTROL"
        ENDIF
```

Note that the only parameters of interest in the AUI AUI_CONTROL opcode 10 are the CSTATUS (which returns the ID of the control, if found), and the the coordinates. Furthermore, the control will be matched if either the starting coordinates (2,8) or the ending coordinates (2,8+XMAX-1) match the actual control.

> The fact that the control can be matched by either the starting or ending coordinates is useful with INFLD controls, whose ending coordinates may be automatically adjusted due to the nature of the control type, and also with controls whose position may be based on some alignment logic, i.e. where you only know the position of the left or right edge.
>
> Note that for debugging purposes, you can determine the internal coordinates and other parameters of all the controls by positioning the mouse on an empty spot in the current dialog or main window and using Control+Shift+Double-Right-Click. This will create and launch a spreadsheet containing the details of all the controls currently defined to A-Shell.

In the above example, as the radio buttons change in the first group, we query the "Impressora" radio button to see if it is selected. If so, we enable the related INFLD controls; if not, we disable them. The logic for this (focusing just on the Copies edit control) would be similar to the following:

```
UPDATE'CONTROL'STATUS:
```

```
! first, query the Impressora radio button, whose ID happens to be RBID(1)
Xcall AUI,AUI_CONTROL, 5, RBID(1), "", 0, "", "", CSTATUS

! if checked (CSTATUS=1) then enable the copies edit, else disable
If CSTATUS = 0 then CSTATE = MBST'DISABLE else CSTATE = MBST'ENABLE
Xcall AUI,AUI_CONTROL, 2, COPIES'ID, "", CSTATE
```

Set Current Tab Pane

Opcode 11 is only used with Tab controls, in order to set (change) the current pane. Each tab in a Tab control is normally defined to send a different exitcode when clicked. Upon receiving the exitcode, the program needs to clear the controls from the current pane, activate the clicked pane, and then recreate the controls that belong on that pane. See the section Tab Control below for a more complete example.

Control Types

This section provides additional notes and examples organized by the various types of controls (static, button, etc.)

Static Text Control

Static text controls are the simplest kind of controls, consisting just of a text string formatted within a rectangular area, with an optional click notification. You may adjust the font and color, but in most cases, will will just want to use the default font and color to get the most uniform, Windows-like look. Since static text controls correspond to simple PRINT statements in the text model, the following PRINT statement:

```
PRINT TAB(10,5);"Customer Name:";
```

would correspond to this static text object:

```
xcall AUI, AUI_CONTROL, 1, ID, "Customer Name:", MBST_ENABLE, MBF_STATIC +
   MBF_LFJUST, "", "", STATUS, 10, 5, 10, 19
```

This probably seems like overkill, but before addressing that, we should first clarify some subtle distinctions between the standard text and static text control implementations.

- Because the text and GUI layers use the same coordinate system, in both cases, the message starts in the same location (row 10, column 5). Also, since we set the ending column of the GUI version to 19, both will occupy the same amount of space. (This fact is important in converting from the fixed pitch to proportional font layouts. But, although the static textcontrol occupies the same amount of space as the text counterpart, some of that space may appear blank (depending on the size of the font and how much difference there is between the fixed and proportional font metrics for this particular string. Typically, lower case characters will take up much less space in a proportional font than in a fixed pitch font, but the reverse may be true with all upper case. So if your text prompts are all upper case, you may need to allow more room (or convert them to lower case, or add the font parameters to the AUI CONTROL call to adjust the font to fit.)
- Because the proportional version of the character string will not appear to take up as much space as the fixed pitch text version, attempts to position adjacent messages based on columnar calculations

will not work very well in the proportional environment. See the section on Preparing to Convert from Text to GUI for further comments on this.

- In the above example,we did not define a command to be associated with clicking on this static text object, but we easily could, in which case the static text object acts like a button. We can also define a tooltip, although this requires that there also be a click command string associated. You can use "VK_NULL" (see Virtual Key Symbolic Names) to define a click command that does nothing, just so you can have a tooltip.)
- Although the static text object exists in a layer above the standard text layer (and thus would obscure anything below it), as a convenience, it may be deleted by the same kinds of PRINT TAB operations that would delete the text version.

If your screen prompts are parameterized (loaded from a file and output using a centralized routine), then converting from the PRINT version to the AUI CONTROL version will be simple. But if you have thousands of literal PRINT statements throughout your programs, you might want to consider one of the following shortcuts:

- PRINT statements may be replaced one-to-one with TPRINT statements, which, when compiled with the /X:2 switch, are equivalent to the AUI CONTROL shown above. That is, every segment or argument to a TPRINT statement will be converted to a separate static text control starting in the position it would have in the fixed pitch environment, and occupying the same frame space (although likely with some trailing space at the right edge of the frame.) When compiled without /X:2, TPRINT statements revert to PRINT.
- PRINT statements may also be replaced with DPRINT statements, which are equivalent to TPRINT except which add the MBF_SUNKEN effect. (In general, this works well for printing data to separate it from other text.)
- SET AUTOTPRINT may be used prior to running a program to automatically treat all PRINT statements like TPRINT. This method provides the fastest way to see how well your screen layouts behave with proportional fonts.
- `PRINT TAB(-10,20); TEXT$; chr(127)` is equivalent to `TPRINT TEXT$`. The advantage of the `PRINT TAB(-10,20)` approach is that it eliminates any compiler dependency (i.e. you could compile under AMOS, or use the same RUN under both AMOS and A-Shell). You would probably need to modify your AMOS terminal driver to filter out the trailing chr(127), or perhaps ideally, to send the necessary ESC sequence so that an ATE client could understand the command just as A-Shell/Windows would have.

All of the above alternatives to the AUI CONTROL class for static text assume left justification. TAB(-10,20) also supports a second argument consisting of T (TPRINT) or D (DPRINT) plus L (left), C (center), or R (right) which give you some more flexibility.

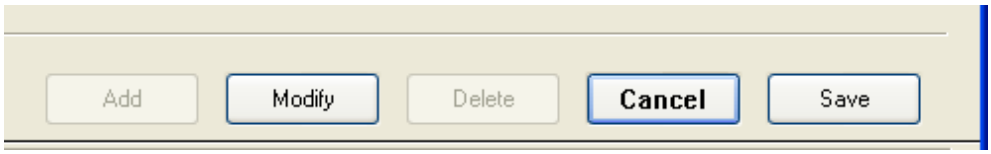
Rectangles and Lines

Rectangles and especially lines can be useful in laying out a screen or window, and are variations of the static text class of controls. See Add for an example.

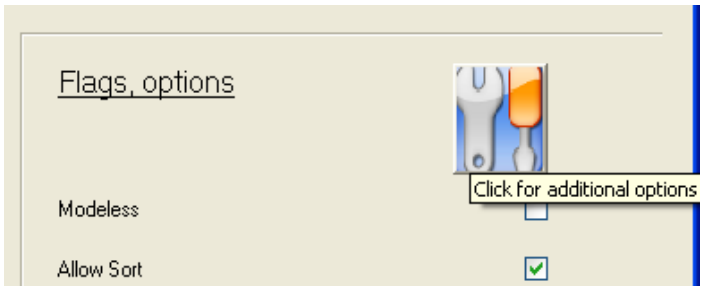
Button Control

After static text controls, buttons are the next simplest. Essentially they can be viewed as nearly equivalent to a static text control except that it gets drawn with a box around it, and you are more obligated to define a command string to go with it. (The command string is option with static text controls, since users do not necessarily demand that all text be clickable.)

Buttons do have a couple of additional features over static text controls. One is that they have a more obviously different appearance when enabled vs. disabled, and when they have the focus, where as static text pretty much appears the same all the time. This example illustrates the different appearance of enabled, disabled, and focused buttons (in the standard XP Visual Theme.)



A second special feature of buttons is that they may display a bitmap (BMP) or icon (ICO) file in place of text. The following example shows a button displaying an icon. Note that when a button displays a BMP or ICO, the image is automatically expanded to fill the button. To help users understand what the button pictures represent, use the *tooltip* parameter to define a tooltip (as shown in this example).



The code to create the icon button in the above screen shot looks like this:

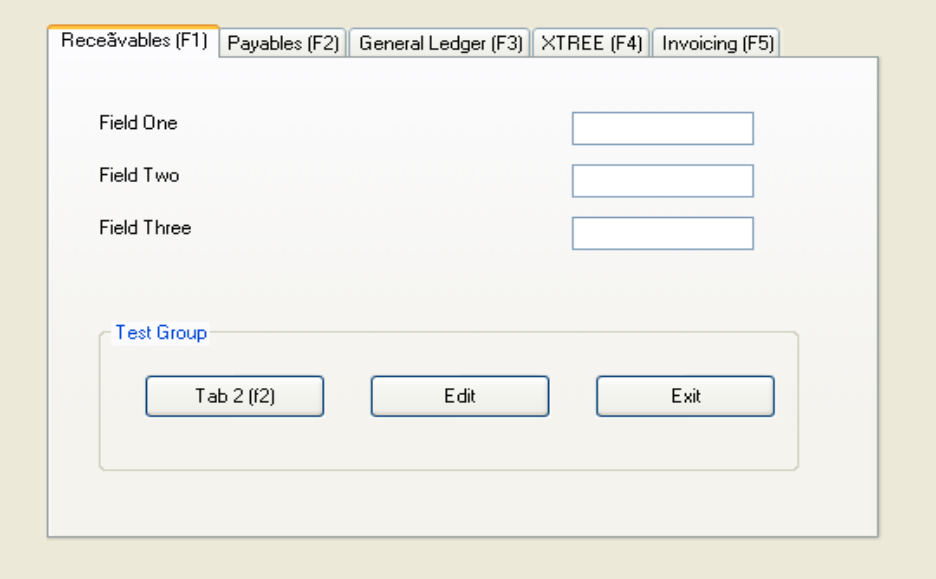
```
xcall AUI,AUI_CONTROL,1,CID,"ctlpan.ico", 0, MBF_BUTTON + MBF_KBD + MBF_ICON,
   "%VK_xF2%", "", STATUS, 3, 72, 4, 75, -1, -1, 0, 0, "", "Click for additional
   options"
```

Bitmap and icon images for buttons may also be loaded from DLL resources. See Button Icon Control below for more details.

Buttons are often grouped together using groupbox controls (which see next).

Groupbox Control

A groupbox is a hollow rectangle with rounded corners used to group a collection of related control. In the following sample, the three buttons are grouped within a groupbox titled “Test Group”.

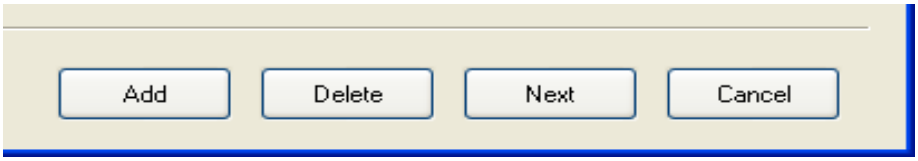


The controls belonging to the groupbox must specify the groupbox *ctlid* as their parent when they are created, and their coordinates are relative to the upper left corner of the groupbox.

Groupboxes are useful in several ways:

- They provide visual organization.
- The fact that the child controls use relative coordinates allows you to adjust the position of the entire group by changing the position of the groupbox.
- The Eventwait class allows the user to move among the buttons within a group, providing a simple way to wait for one of a group of option buttons.
- Grouping controls together using a groupbox allows them all to be deleted in a single step by deleting the groupbox.
- Groupboxes are needed to define more than one set of radio buttons.

Groupboxes do not have to be visible to offer all of the above features (except for the first one). Also, the child controls do not have to be confined geometrically within the groupbox. The groupbox above would work just as well if the box were invisible, and even if it was reduced to a one by one block in the upper left corner. Also, child controls may be placed on top of the border (whether visible or not). These facts come in handy when you want to have a group of buttons for control purposes, but you don't want to waste any screen space for it. For example, you can make the groupbox invisible and one row high, then put a row of buttons on that same row. Or, another common arrangement is to put a row of buttons at the bottom of a window, with just a line above it:

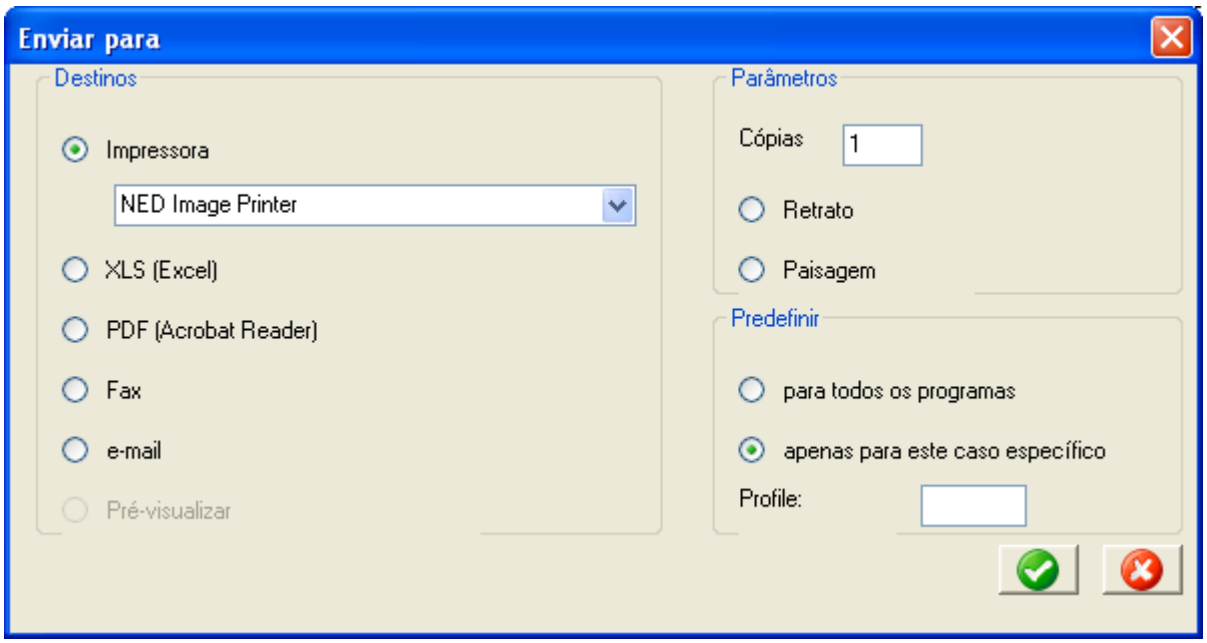


In the example above, an invisible group was probably created, occupying the bottom two rows of the screen. On top of the invisible group, a single line was drawn (see Rectangles and Lines below), along with the buttons below it. (BTNMNU.SBX can be used to create groups of buttons like this.)

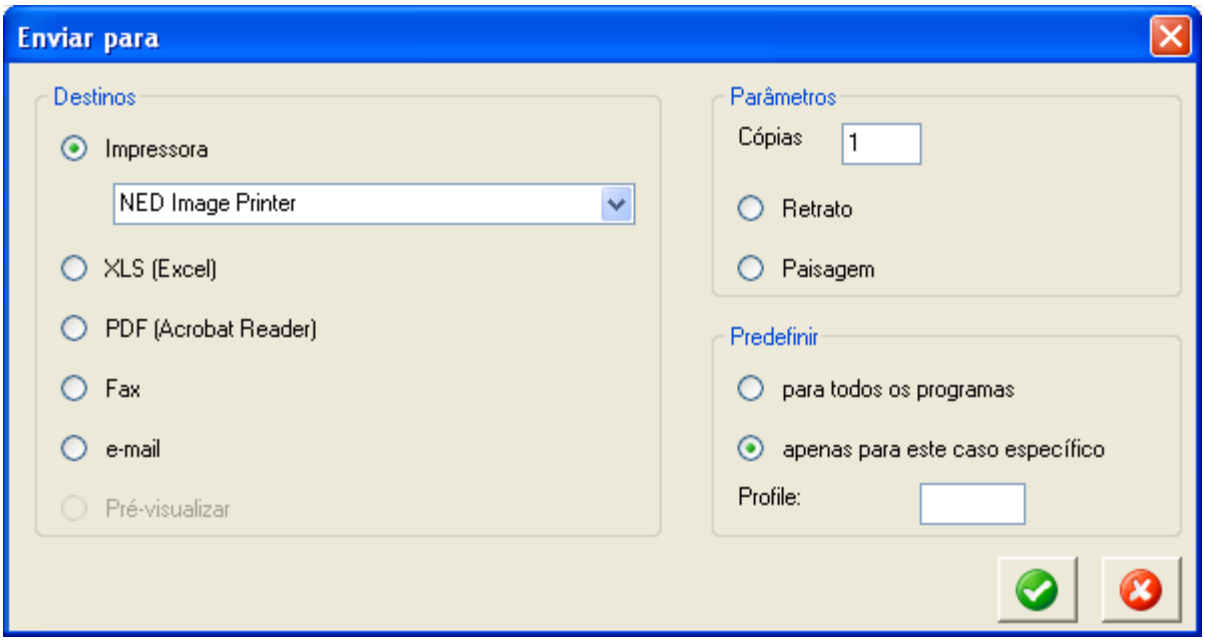
A groupbox one row high still contains 4 sides, so if you want a single line like in the above example, you have to use the rectangle technique.

Groupbox + MBF_ALTPOS

In the standard groupbox layout, the top border is positioned at the very top of row 1, and the bottom border is about in the middle of the bottom row. This is not always ideal, particularly when you want to put controls on every row within the groupbox. The illustration below shows what can happen. In each of the three groupboxes, the first control within is on row 2 (relative to the groupbox), and the last control is on the bottom line of the groupbox. For example, the last groupbox (Predefinir) is four rows high, with controls on rows two, three, and four. The "profile" text control and edit box on row four share the row with the bottom border of the box.. The problem is that we have plenty of room between the top border and the controls in row two, but almost no room between the bottom border and the controls on the bottom row. In fact, the parts of the border are being obscured by the controls "Pré-visualizar", "Paisagem", and "Profile". Also, the buttons on the bottom are not centered vertically very well in the space between the bottom of the dialog and the bottom of the groupbox. (Note that the difference between this and the previous example above which shows a set of four buttons below a horizontal line, is that the horizontal line is positioned vertically near the top of the row, whereas the whereas the bottom border of the groupbox is closer to the bottom.



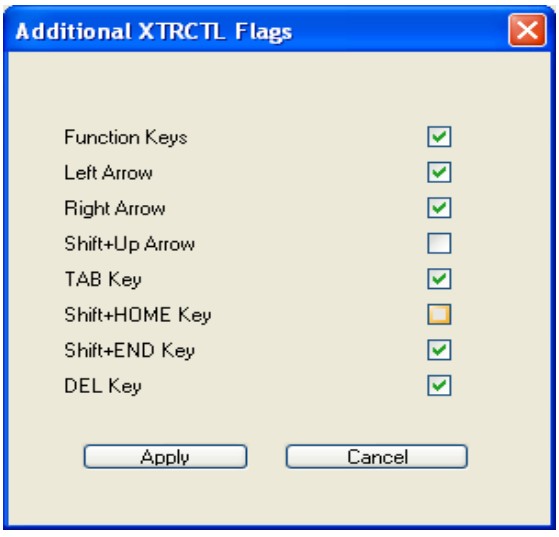
Adding the MBF_ALTPOS option to both the groupboxes and the buttons at the bottom shifts the spacing and sizing of the groupboxes and buttons, resulting the in the example below. The top border of the groupboxes has been moved down by about 1/3 of a row, and the bottom border has been down slightly less than that, resulting in more even vertical spacing between the borders and the controls within. Also, for the buttons, MBF_ALTPOS causes them to take up the entire row height, including the inter-row (aka "leading") area, and to be shifted down to take advantage of some of the margin which is added to the bottom of dialogs.



Checkbox Control

Checkboxes are the Windows GUI equivalent of yes/no questions.

Typically several checkboxes are presented as a group, either within a groupbox, or in a dialog as in this example below:



When the group of checkboxes has no other input fields which require realtime support by the application, the easiest way to implement them may be to just display them, then use the EVENTWAIT class to wait for a button action (Apply or Cancel in the above dialog), then query the buttons to see which ones are checked. In the Windows environment, you don't even need to query them, since one of the parameters passed to AUI CONTROL to create the checkbox is the actual variable which gets updated automatically as the box is checked/unchecked. But in the ATE environment, direct linkage between the checkbox on the screen and the

control variable is not possible, so the technique of querying the checkboxes is more universal. See the section below for sample code showing how to create a query a group of checkboxes like is shown above.

The other way to manage checkboxes is INFLD, which makes them act like yes/no fields from the point of view of the application. To turn an existing INFLD yes/no field into a checkbox, just add the TYPE code **||c**. In this case, you'll need to process exitcodes like for any other field in order to allow the user to move up and down or among the fields.

Note that although INFLD does a reasonable job of hiding the idiosyncrasies of checkboxes from the application (making them act like yes/no fields), it is hard to overlook the fact that unlike other field types, with a checkbox, the label or prompt is actually part of the control, so you need to specify it to INFLD if you want it to create the checkbox with the text. To do so, put the label text in the SETDEF parameter, and you must set the field width to accommodate the text plus the checkbox. However, as a convenience to programs that are used to displaying the field prompts in one operation, and editing the fields in another, when INFLD is given a standard one-character-wide yes/no setup with the ||c option added, it will scan the existing controls to see if this matches the left or right edge of an existing checkbox, and if so, uses the exiting one rather than creating a new one. This allows you to create the checkboxes as part of your screen layout operation, but then edit them as if they were simply yes/no fields.

Checkbox Alignment, Justification

There are two parts to a checkbox control (the text label and the checkbox itself), and six ways to arrange them, as shown here (from the TSTCBZ sample program):

The checkboxes in the upper group (purple text) were created with these AUI_CONTROLAUI_CONTROL calls:

```
xcall AUI, AUI_CONTROL, OP, ID(1), "Left Text; Left Justify",  MBST_ENABLE,
   MBF_CHKBOX + MBF_LFTEXT + MBF_LFJUST, CMD$, CB1,  STATUS, 2, 2, 2, 24, 67, -2,
   0, 0, "",  "", GRPID1

xcall AUI, AUI_CONTROL, OP, ID(2), "Left Text; Right Justify",  MBST_ENABLE,
   MBF_CHKBOX + MBF_LFTEXT+MBF_RTJUST, CMD$, CB2,  STATUS, 3, 2, 3, 24, 67, -2,
   0, 0, "",  "", GRPID1

xcall AUI, AUI_CONTROL, OP, ID(3), "Right Text; Left Justify",  MBST_ENABLE,
   MBF_CHKBOX + MBF_LFJUST, CMD$, CB3,  STATUS, 5, 2, 5, 24, 67, -2,  0, 0, "",
   "", GRPID1

xcall AUI, AUI_CONTROL, OP, ID(4), "Right Text; Right Justify",  MBST_ENABLE,
   MBF_CHKBOX + MBF_RTJUST, CMD$, CB4,  STATUS, 6, 2, 6, 24, 67, -2,  0, 0, "",
   "", GRPID1

xcall AUI, AUI_CONTROL, OP, ID(5), "Left Text; Center Justify",  MBST_ENABLE,
   MBF_CHKBOX + MBF_LFTEXT, CMD$, CB5,  STATUS, 8, 2, 8, 24, 67, -2,  0, 0, "",
   "", GRPID1
```

```
xcall AUI, AUI_CONTROL, OP, ID(6), "Right Text; Center Justify",  MBST_ENABLE,
   MBF_CHKBOX, CMD$, CB6,  STATUS, 9, 2, 9, 24, 67, -2,  0, 0, "",  "", GRPID1
```

The checkboxes in the lower group (blue text) were created with INFLD (which only supports four of the six possible alignments):

```
xcall INFLD, 2, 2, 23, 0, "||c", CB$(1), INXCTL, GRPID2+1, 2, EXITCODE, 0, 0, -1, -
   1, -1, "Left Text; Left Justify"

xcall INFLD, 3, 2, 23, 0, "||c|J", CB$(2), INXCTL, GRPID2+1, 2, EXITCODE, 0, 0, -1,
   -1, -1, "Left Text; Right Justify"

xcall INFLD, 5, 2, 23, 0, "||cR", CB$(1), INXCTL, GRPID2+1, 2, EXITCODE, 0, 0, -1,
   -1, -1, "Right Text; Left Justify"

xcall INFLD, 6, 2, 23, 0, "||cR|J", CB$(2), INXCTL, GRPID2+1, 2, EXITCODE, 0, 0, -
   1, -1, -1, "Right Text; Right Justify"
```

TYPE code legend: **||c** is for checkbox; **R** reverses the normal alignment (putting the checkbox on the left and the text on the right), and **|J** right justifies the text. (**|J** is the same as setting SBR=INFLDCBRJ but has the advantage of being easy to change from one field to the next.)

INFLD uses the SETDEF parameter for specifying the text associated with checkboxes.

Radio button Control

Radio buttons are similar to checkboxes, except that within a group, only one can be set at a time, as shown in the example below:

This is a typical way of presenting a choice among several mutually exclusive options, and can be implemented using either of the techniques described above for checkboxes. (If using INFLD, use **||r** rather than **||c** to create a radio button field.) However, despite the obvious parallel to checkboxes, radio buttons do not correspond neatly to any legacy data entry INFLD construct, because it breaks the normal one data field to one input field correspondence.

For example, consider a payroll application that needs to know the marital status, with the choices being single, married filing together and married filing separately. Since this information is going to be stored as a single datum (probably coded as a number), it would be easiest to just prompt for the code number (1-3) while displaying a menu. To present this as a series of radio buttons, we need more screen real estate and logic to check each of the fields to see which one was selected. So, while they may look neat, they may not be that practical to implement, especially when migrating from, or trying to preserve compatibility with a text

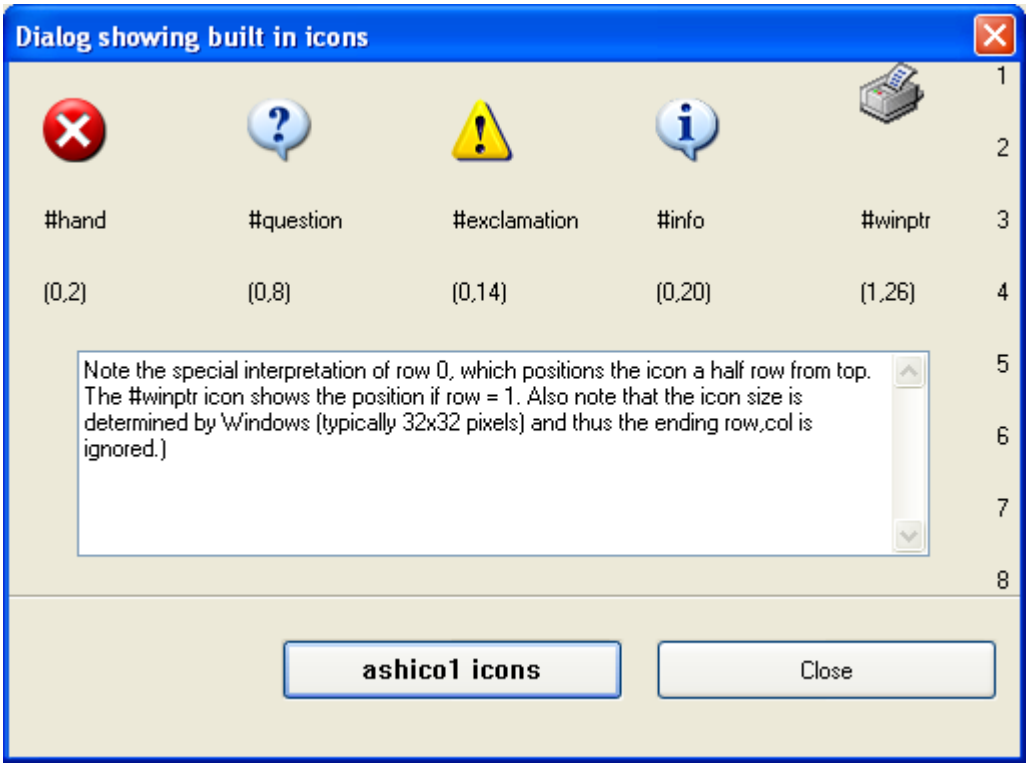
implementation of the same application. A much more natural, and legacy-friendly way to present such a choice is via a combo box.

Icon Control

Icon controls may be created as a variation of either the STATIC or BUTTON control types. In general you would use the STATIC type for icons that are just decorative, and the BUTTON type for icons meant to be clicked on. The following two subsections deal with the specifics of each type:

Static Icon Control

The STATIC version of the icon control gives you a non-scaled icon with no border, which is appropriate for displaying as a sort of logo in a dialog. This example shows a dialog with several icons, just for illustration purposes:**Error! Bookmark not defined.**


Dialog showing built in icons
#hand
#question
#exclamation
#info
#winptr
(0,2)
(0,8)
(0,14)
(0,20)
(1,26)
Note the special interpretation of row 0, which positions the icon a half row from top. The #winptr icon shows the position if row = 1. Also note that the icon size is determined by Windows (typically 32x32 pixels) and thus the ending row,col is ignored.)
ashico1 icons
Close

Static icons are created by setting ctype to *MBF_STATIC* + *MBF_ICON*, and ctext to the resource name of the icon. There are three possible sources of icon resources, each with its own syntax:

- System icon resources. These are embedded in Windows, and are referenced with a name starting with "#". As of build 919, the only ones available are shown in the example above and listed in the table below (the names are not case sensitive):

ICONAMES	Description
#HAND	Same as the MBICON'stop icon in MSGBOX. In XP, this appears as an X in a red circle.

ICONAMES	Description
#QUESTION	Same as the MBICON'question icon in MSGBOX.
#EXCLAMATION	Same as the MBICON'exclamation in MSGBOX. In XP, this is a yellow triangle with an exclamation mark in it.
#INFO	Same as the MBICON'info icon in MSGBOX. In XP, this is an "i" in a white caption bubble.
#WINPTR	A generic printer icon.

For example:

```
xcall AUI, AUI_CONTROL, 1, ctlid, "#question", MBST_ENABLE, MBF_STATIC + MBF_ICON,
   "", "", cstatus, srow, scol, erow, ecol2
```

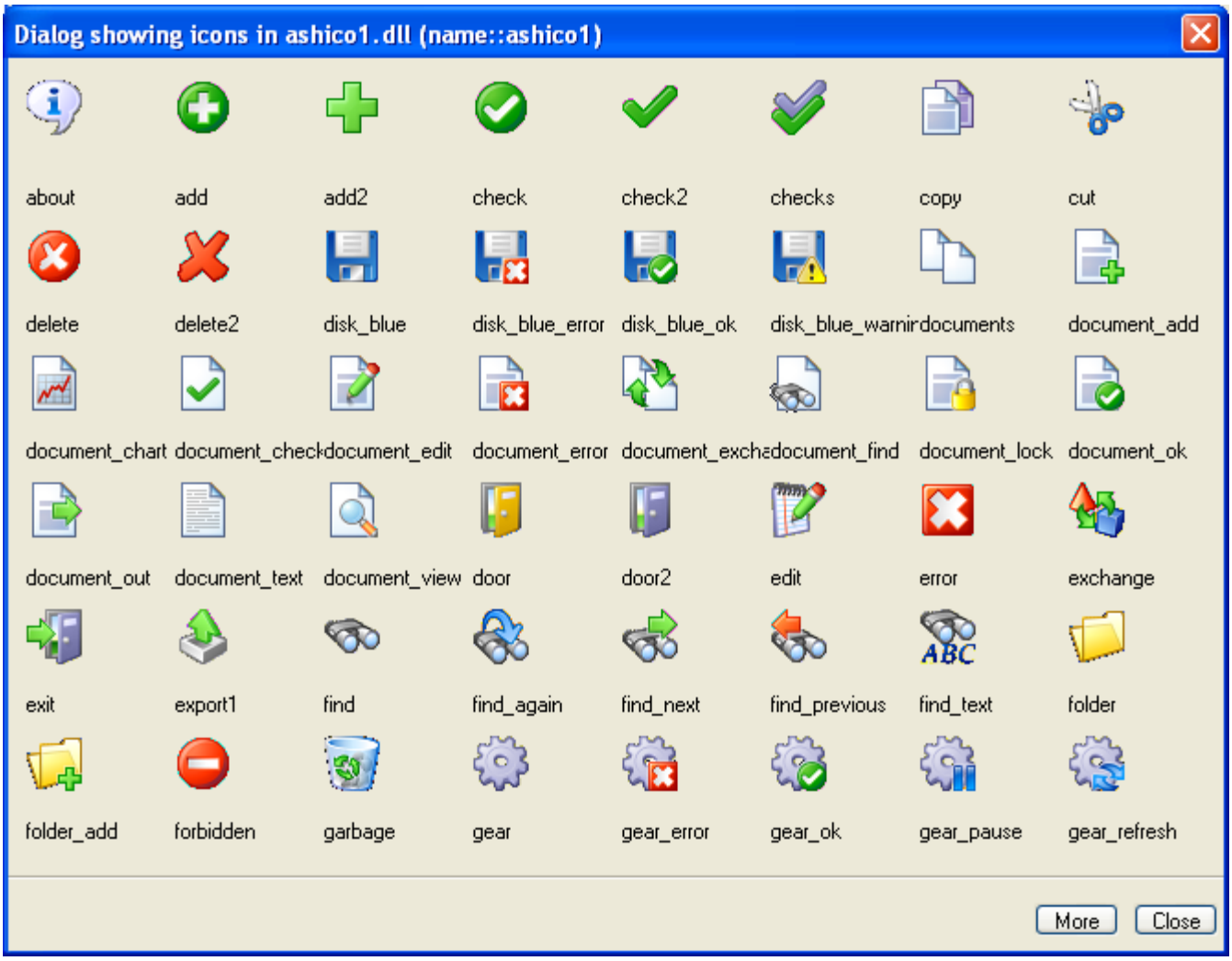
- A-Shell internal icon resources. These are icons embedded in the ashw32.exe executable, and consist of icons used internally by A-Shell, plus any third-party application icons that have been sent to us for embedded in ashw32. The syntax for these is just a string name, without the "#", such as "ACORNICON". (If we embedded an icon specially for you, we will tell you the name.)
- Icons loaded from external DLLs. In this case the syntax for *ctext* is "**resourcename::modulename**" where **resourcename** is a string name, such as "stop", and modulename is the name of the DLL (or other loadable module). For DLL's, you do not need to specify the DLL extension, but the DLL file must be located in the same directory as the copy of ashw32.exe that is being launched, or else in the WINDOWS or SYSTEM32 directory. For example, an icon source named "document_check" in the ashico1.dll library would be referenced as "`document_check::ashico1`".

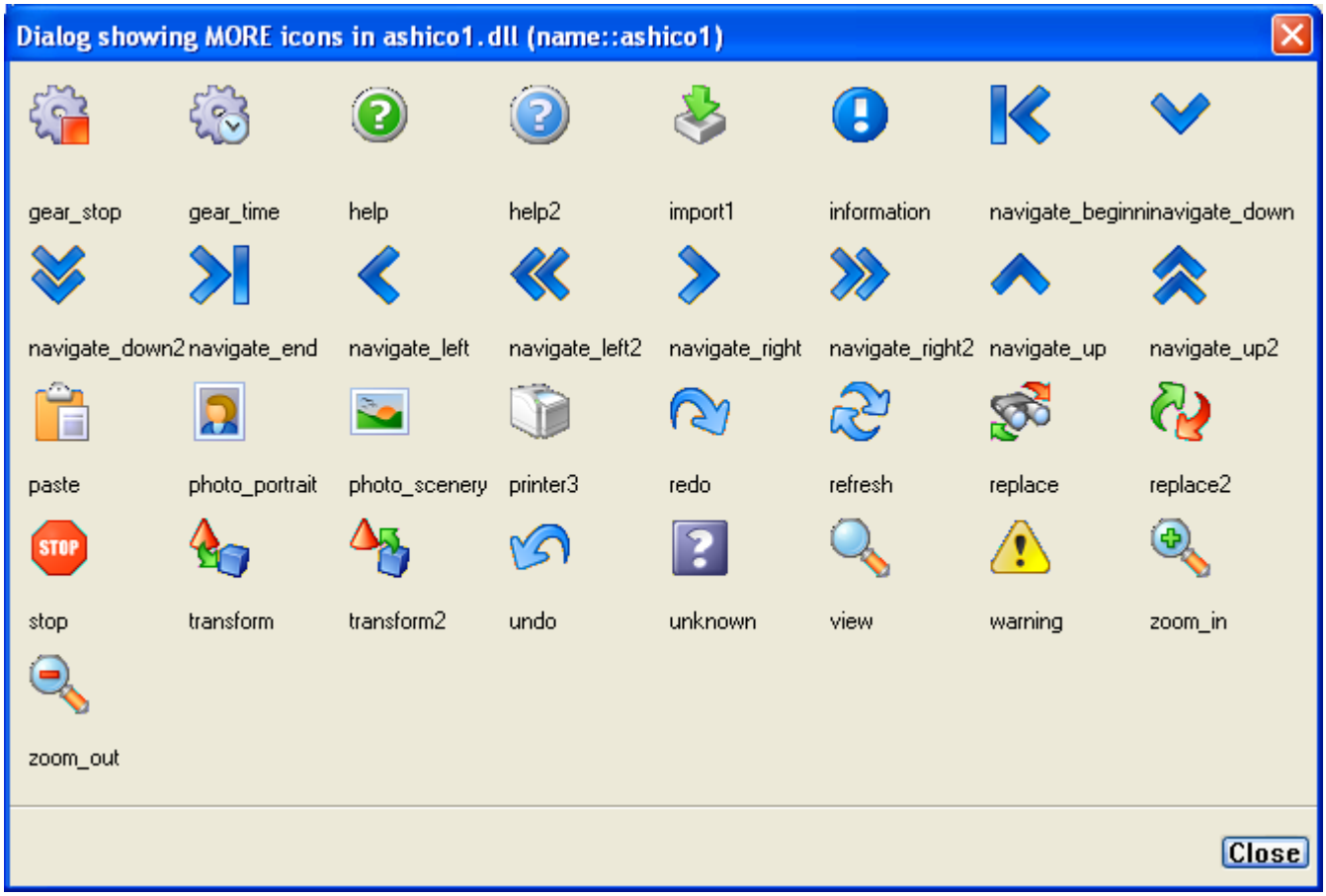
Static icons are not scaled to the size of the control. Instead, they are fixed by Windows, according to characteristics of the desktop environment. In almost all cases, this means they are 32 x 32 pixels. The two consequences of this are that the *erow* and *ecol* parameters do not matter (by convention they should probably be set to match the *srow* and *scol* parameters), and that you need to allow for adequate space for the icons in cases where the number of pixels in the window is smaller than what you designed with. For example, if the window size is about 800 x 500, a 32 x 32 icon will occupy about 1.5 rows in height and 3.2 columns in width. But the same icon in a window of 1200 x 1000 will occupy less than 1 row in height and about 2 columns in width.

Static icons in dialogs interpret a starting row of 0 as a special flag to position it with about ½ row of top margin below the title bar of the dialog. This is just about right for the typical case. (Row 1 puts the icon right up against the title bar, and row 2 would waste too much space.) The sample dialog showing the five built-in icons above illustrates this. The first four icons are positioned with *srow* = 0, while the last one uses *srow* = 1.

Icon Library

MicroSabio provides an icon library in the file ashico1.dll, which contains the icons shown and listed below. These have been purchased by MicroSabio so they can be used, royalty-free, by A-Shell developers. While these icons are free to use, most icons are copyrighted and/or licensed and should not be used without the explicit permission of their owner. Icon collections, such as MicroSabio's, are available for purchase from many icon vendors and developers. The library may be displayed with the sample program ICODLG.


Dialog showing icons in ashico1.dll (name::ashico1)
about
add
add2
check
check2
checks
copy
cut
delete
delete2
disk_blue
disk_blue_error
disk_blue_ok
disk_blue_warnir
documents
document_add
document_chart
document_check
document_edit
document_error
document_excha
document_find
document_lock
document_ok
document_out
document_text
document_view
door
door2
edit
error
exchange
exit
export1
find
find_again
find_next
find_previous
find_text
folder
folder_add
forbidden
garbage
gear
gear_error
gear_ok
gear_pause
gear_refresh
More
Close

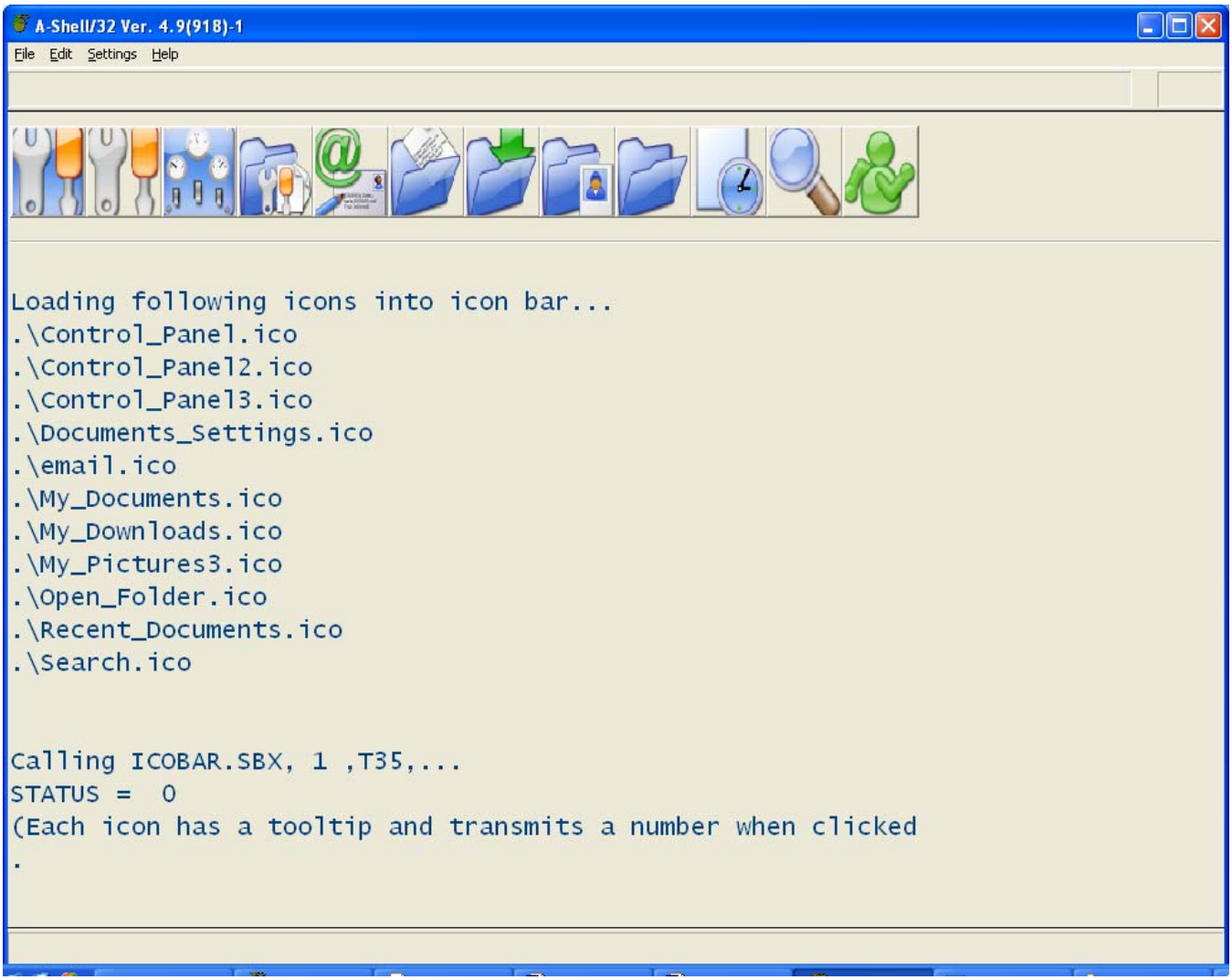


Here is a listing of the icons' names, since they are hard to read in the size shown above.

about	document_chart	exit	gear_stop	paste
add	document_check	export1t	gear_time	photo_portrait
add2	document_edit	find	help	photo_scenery
check	document_error	find_again	help2	printer3
check2	document_exchange	find_next	import1	redo
checks	document_find	find_previous	information	refresh
copy	document_lock	find_text	navigate_beginning	replace
cut	document_ok	folder	navigate_down	replace2
delete	document_out	folder_add	navigate_down2	stop
delete2	document_text	forbidden	navigate_end	transform
disk_blue	document_view	fgarbage	navigate_left	transform2
disk_blue_error	door	gear	navigate_left2	undo
disk_blue_ok	door2	gear_error	navigate_right	unknown
disk_blue_warn	edit	gear_ok	navigate_right2	view
documents	error	gear_pause	navigate_up	warning
document_add	exchange	gear_refresh	navigate_up2	zoom_in

Button Icon Control

Button icon controls are typically used for icons intended to be clicked on to perform some action, as in this sample (TSTICB, which uses the subroutine ICOBAR.SBX):



Button icons use the *ctype* combination *MBF_BUTTON* + *MBF_ICON*. (In most cases, you would also add the *MBF_KBD* type to specify a string to be sent when the button is clicked.)

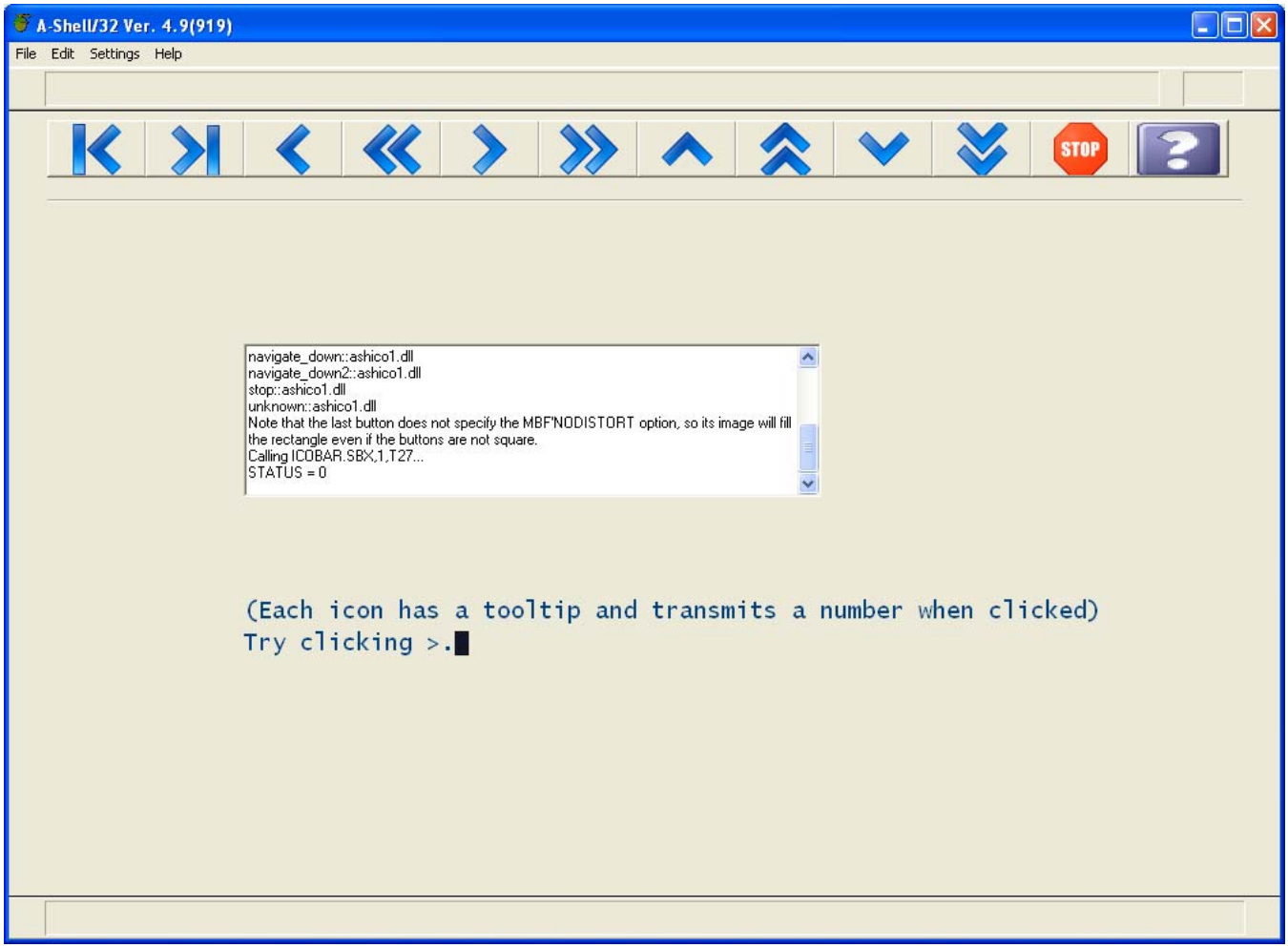
Button icons can be loaded individually from .ICO files, or via resource names from a DLL. (As of build 919, they do not support the Windows internal icon resources such as "#question".) To load an icon into a button from an ICO file, just specify the ICO filespec in the *ctext* parameter, either using AMOS or native syntax, for example:

```
xcall AUI,AUI_CONTROL, 1, id, ".\control panel.ico", mbst_enable, mbf_button +
   mbf_icon, cmd$, "",  status, srow, scol, erow, ecol, -2, -2,  0, 0, "",  "",
   grpid1
```

To use an icon resource from a DLL, use the same syntax as for static icons, i.e.:

```
xcall AUI, AUI_CONTROL, 1, id, "navigate_down::ashico1.dll", mbst_enable,
   mbf_button + mbf_icon, cmd$, "", status, srow, scol, erow, ecol, -2, -2, 0, 0,
   "",  "", grpid1
```

Unlike static icons, button icons are scaled to fit the button. However, this can sometimes result in a distorted look if the button is not square. (In the example above, the buttons are slightly taller than they are wide, which results in a slightly "squeezed" look.) To prevent that kind of distortion, you can add the flag *MBF_NODISTORT* (4194304), which forces it to retain the original square aspect ratio. The example below illustrates some icons from the ashico1.dll displayed as buttons with the *MBF_NODISTORT* flag (all except the last icon, which illustrates what it would look like without the *MBF_NODISTORT*.:


A-Shell/32 Ver. 4.9(919)
File Edit Settings Help
STOP
navigate_down::ashico1.dll
navigate_down2::ashico1.dll
stop::ashico1.dll
unknown::ashico1.dll
Note that the last button does not specify the MBF'NODISTORT option, so its image will fill the rectangle even if the buttons are not square.
Calling ICOBAR.SBX,1,T27...
STATUS = 0
(Each icon has a tooltip and transmits a number when clicked)
Try clicking >.

Also see the sample dialogs under Groupbox + MBF_ALTPOS for a good example of how using an icon on a button can be much easier to understand than text in a foreign language (which may be how your application appears to even native users.)

Icons: Static vs. Button

The distinctions between icons implemented as static controls (*MBF_STATIC* + *MBF_SSICON*) and those implemented as buttons (*MBF_BUTTON* + *MBF_ICON*) are summarized here:

- Icons on buttons are scaled to the size of the button, whereas icons in static controls display in their "natural" size (which in most cases is 32x32 pixels, although on an extremely low res or high res screen, or if you tinker with the Windows system parameters, it may be as small as 16x16 or as big as 64x64). Although the icon size (for static controls) is not affected by the *erow* and *ecol* parameters, we recommend setting them equal to *srow* and *scol*.
- Only buttons can load icons from individual files, whereas only static controls can load them from the internal system resources (e.g. "#INFO"). Either type of icon can load the image from a DLL.
- Static controls, including icons, may have click actions defined, but they cannot get the focus while waiting in the EVENTWAIT routine.

Bitmap Control

Bitmap controls are very similar to icon controls, and come in the same two flavors – static (*MBF_STATIC* + *MBF_BITMAP*) and button (*MBF_BUTTON* + *MBF_BITMAP*). The main differences between bitmaps and icons are:

- Icons (whether stored in an ICO file or in a resource) typically contain variations of themselves with different color depths and grid sizes, from which Windows will usually select the most appropriate one. Bitmaps, on the other hand, are stored with only one size, and are typically converted internally to a format appropriate for the display device.
- Icons are square, whereas bitmaps can be in any rectangular shape.
- The *MBF_NODISTORT* flag only applies (as of build 919) to icons.

Edit Control (INFLD)

An edit control is a rectangle in which text may be edited. Most commonly they are used to input just a single field, limited to one line of text and a certain number of characters. Although it is theoretically possible to use the AUI CONTROL class to create an edit box (using the *winclass* parameter), as a practical matter, edit boxes are handled by INFLD, which takes care of the details of creating and managing the edit control and providing a single interface to the application which works in both text and GUI modes. Refer to the INFLD Reference for full details on using INFLD with edit and other GUI controls. For convenience, the GUI-related TYPE codes are excerpted here. See Groupbox + MBF_ALTPOS, Checkbox Alignment, Justification, Date Picker Control (INFLD), and Time Picker Control (INFLD) for information about other INFLD control types.

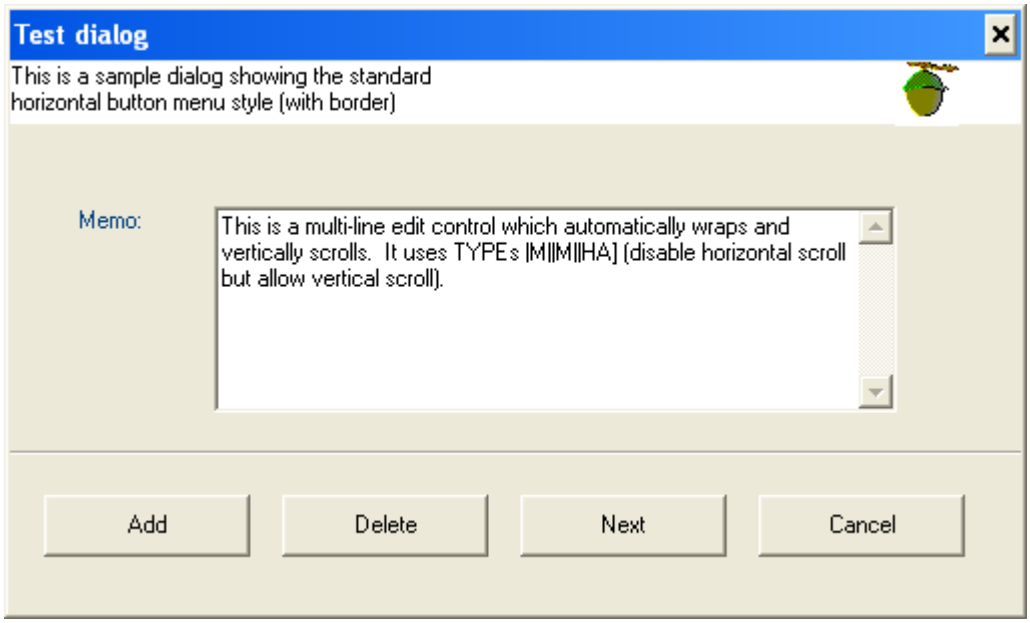
INFLD supports several TYPE codes which activate various graphic user interface features within the A-Shell/Windows (or ATE) version of INFLD.

When in GUI mode, INFLD typically assumes the form of a Windows edit control when editing, and a static text control (with sunken edges) when not currently being edited. For date and time fields, a date or time “picker” (e.g. calendar /clock) control may be used, and for fields with certain kinds of SETDEF lists, a combo box may be used. See

GUI-related Codes in the **INFLD.SBR** topic for more details.

Multi-line Edit Control (INFLD)

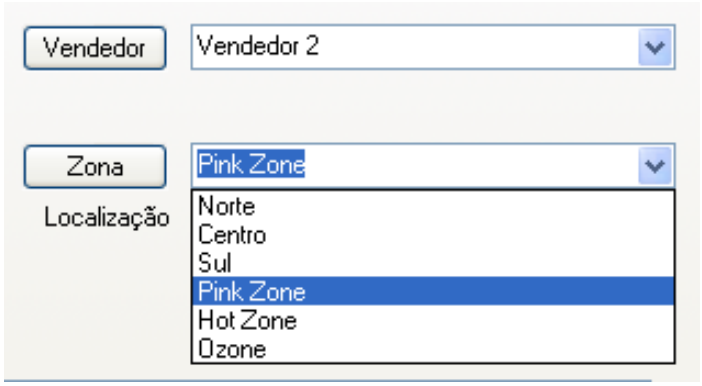
A multi-line edit control is a special case of the standard edit control. See types |**M** and ||**M** and ||**H** in the table above for notes on implementing one. Note that although this is a convenient way of editing and displaying a free format block of text, it is currently only suitable for up to about 1800 characters, and doesn't begin to match the capabilities of INMEMO. Here's a simple example:


Test dialog
This is a sample dialog showing the standard
horizontal button menu style (with border)
Memo:
This is a multi-line edit control which automatically wraps and vertically scrolls. It uses TYPEs |M||M||HA] (disable horizontal scroll but allow vertical scroll).
Add
Delete
Next
Cancel

The SCRSTS.SBX subroutine uses a multiline edit control in display-only mode to implement a scrolling status window. It is available, along with sample programs, in the A-Shell Open Source Library.

Combo Box Control (INFLD)

A combo box is another compound control type, made up of an edit control and a drop-down list box. As with other variations of edit controls, they are managed by INFLD rather than by the AUI CONTROL class. The example below shows two combo boxes, one in the display state and the other being edited and with the drop-down list extended.



Thanks go to Jorge Tavares for the example above, which besides illustrating combo boxes, also illustrates the use of a button as the label or prompt associated with the combo box. In this case, clicking on the button pops up a dialog, allowing the qualified user to edit the list of items available to be chosen in the drop down list.

There are a number of variations of combo boxes. In the default state, you can freely type in the edit box. (Whether or not INFLD allows you to exit the field after entering a value not in the list depends on whether you add the TYPE ||**s**, or include a wildcard "*" in your SETDEF list.) As with other INFLD fields, TYPE **O** may be added to make the field optional, i.e. to allow a blank entry. If you want to limit the operator to just selecting values from the list (either by displaying it and clicking on an option, or by typing one or more characters and using the auto-selection logic), then add TYPE code ||**S**.

The width of the drop-down list will be increased beyond the XMAX value, if necessary to accommodate the longest choice (as specified in the SETDEF parameter). This allows you to have longer, more descriptive choices in the drop-down list than the field allows for. A good example of this would be a 2-character state abbreviation field, which you can now link to a list of choices such as ",CA California,MN Minnesota,," etc. In this case the user will be able to see the full descriptions, but the data returned to the program will be limited by XMAX (i.e. to just the 2 character state abbreviation).

There is no set maximum to the number of characters (or number of choices) that can be displayed in the drop-down list, but once it gets beyond a few thousand bytes or a few hundred entries, you will probably want to consider another method, such as XTREE, or perhaps a Self Service Combo Box.

Text Mode Compatibility

INFLD supports most of the combo box-related TYPE codes even in text mode, only in that case, instead of a combo box like shown above, you get a normal looking field, with a display of the list of options along the bottom line of the screen. (The display can be turned off with TYPE **s**.) This is a good example of a GUI enhancement that you get with INFLD without any programming changes, simply by activating GUI mode (with |**G**).

Coded Lists

Often, if a list has only a limited number of items, and especially if the description of those items might be lengthy, it makes more data processing sense to store the selected item as a coded number, rather than as text. For example, the program shown above actually stores the choices for the Zona field as a B,1 value, maintaining a table which translates the numeric codes into the descriptions which are seen above. Rather than having to do this work separately, you can let INFLD manage the table by adding TYPE ||**L**, and specifying SETDEF as a list of pairs, i.e.

```
SETDEF="01, Norte, 02, Centro, 03, Sul, 04, Pink Zone, 05, Hot Zone, 06, Ozone, , "
```

In this case, INFLD displays just the descriptions, but returns to the program the numeric codes (in other words, letting the user and the programmer have it their own way.)

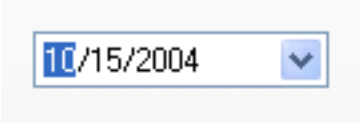
A sub-variation of the above causes INFLD to return both parts of the pair, i.e. "04,Pink Zone" if TYPE ||**l** (lower case L) is used instead of ||**L**.

Self Service Combo Box

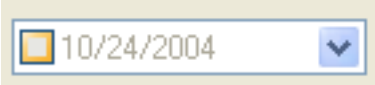
Another variation of a combo box is where you want to present the user with the appearance of there being a combo box to select from, but you don't want to specify the choices in advance. One reason might be that the choices are actually all the records in a file, and rather than fetch them in advance, you want to allow the user to enter a couple of characters to limit the number of choices, then presenting them in an XTREE control (which is equipped to handle thousands of entries, unlike the combo box). To get this effect, specify SETDEF="..." (3 dots). INFLD will then return EXITCODE 29 to the application when the user clicks on the down arrow button or hits a key that would normally display the list (including the down arrow key)

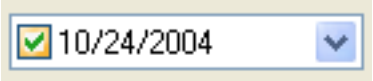
Date Picker Control (INFLD)

A date picker is a special kind of edit control designed only for the input of dates. It is created automatically by INFLD when called on to edit a date field (TYPE **D**) in GUI mode (|**G**). (It is not activated in the "lesser GUI mode", which you get with TYPE |**g**.) In the first example below, the field is being edited, but unlike with a normal field, the control selects one sub-field (month, day, or year) to edit at a time. You can either type digits, or use the up and down arrows to increment/decrement the value. Use the right arrow to move to the next sub-field.

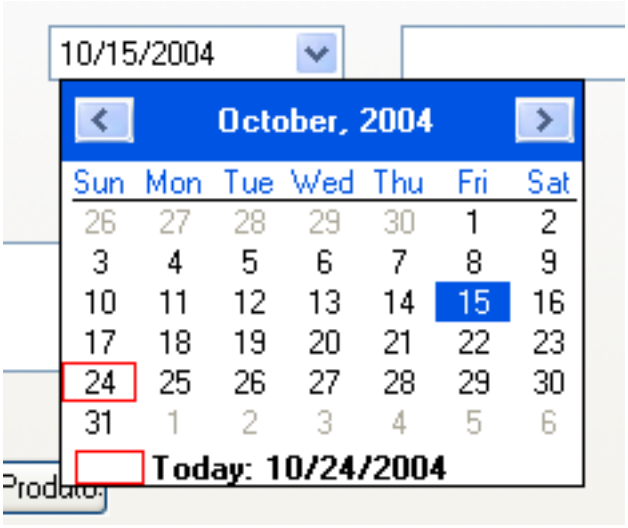


If the date is optional (XMIN=0), a checkbox will appear. If unchecked, the date is grayed out and unavailable. To change it, first check the checkbox:





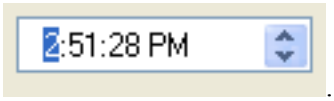
To display the calendar, click on the down arrow button, or hit ALT-down arrow:



The format used by the date picker for editing and display is based on the localization settings of the client PC. But the format returned to the application is based on the INFLD TYPE codes and XMAX value. With XMAX=6, only 2 digits of the year are turned. With XMAX=8, the entire CCYY is returned. TYPE **D** uses Amercan format (MM/DD/{CC}YY), TYPE **U** forces European (DD/MM/{CC}YY) format, and TYPE **u** selects the format from the language definition file (e.g. ENGLSH.LDF[1,6]). TYPE **>** forces the date to be returned in CCYYMMDD format (most convenient for sorting, independent of country format).

Time Picker Control (INFLD)

A time picker is similar to a Date Picker Control except for time rather than dates. (Use INFLD TYPE **t** or **||t**.) As with the date picker, the editor selects one sub-field at a time for editing (hour, minute, second) and you can edit it either by typing or by clicking on the up or down arrows.



The format used by the control for editing and display is based on the localization settings for the workstation, but the format in which the field is returned is determined by the TYPE code and XMAX values. With TYPE **t**, the time is returned in military (24 hour) format. With TYPE **||t**, the time is returned in 12 hour format with AM/PM appended. In both cases, setting XMAX to 5 will eliminate the seconds from the returned time string. (Otherwise 8 is the normal XMAX value.)

Modal Dialog Box

A "modal" dialog (*MBF_DIALOG*) is a popup window that holds on to the keyboard focus, making it impossible to type in any other window owned by the current application, until the modal dialog is closed. You can, however, drag the dialog around, access the main A-Shell menu bar, and even pop up nested modal dialogs. They are quite handy when you want to interrupt the context of the current operation to display or input a collection of related fields.

When creating a dialog box, you should save its *ctlid* parameter so that it can be used later with opcode 3 to delete the dialog box. Deleting the dialog box will automatically delete the controls within it.

Once a modal dialog is opened with A-Shell, any subsequently created controls (buttons, static text, edit prompts, etc.) will be owned by the dialog box, and their coordinates will be taken as relative to the dialog box. (In other words, the *parentid* parameter for will automatically be set to the dialog box ID.) However, it is probably still a good idea for you to manually set the parentid, as it probably will make your programs easier to follow, and will be indispensible if you want to implement a text version of a dialog using MSBOXX. Ordinary text output (e.g. PRINT statements) will ignore the dialog box and appear in the main A-Shell window. (In other words, you can only place Windows control-objects within these dialog boxes.)

Typically, a dialog box should have one or more buttons which notify the program (via a keyboard sequence) to close the dialog (for example, "OK", and "CANCEL"). You may also include an "X" in the upper right corner of the dialog by adding the *MBF_SYSMENU* flag to *ctype*, which the user can click on to close the dialog.

Note that clicking on the "X" to close a dialog does not directly close it. Instead, it sends an ESC character to the keyboard buffer, which the dialog's input routine should process by closing the dialog (with opcode 3). To simplify programming, if the dialog also has a CANCEL button, you might as well program it to send ESC as well, so that pressing on the CANCEL button or "X" has the same effect. Also note that you are not obligated to close the dialog just because the user clicked on either of these buttons. If there is a problem needing resolution by the user, you can ignore the clicks or pop up a message box (another modal dialog) to tell the user what they must do before closing the dialog.

There are several sample programs which contain dialogs, including XTRA2, XTRA3, and TABDLG. See the sample screen shot in the Tab Control Tab Control Example below for an example of a dialog.

Progress Bar Control

A progress bar (*MBF_PROGRESS*) is similar to a hollow static rectangle which you can progressively fill during the course of an operation to give a visual indication of the percentage of the job completed.

The control is similar to a static text control, except that you must specify the *MBF_PROGRESS* flag instead of *MBF_STATIC*, and currently the colors are ignored.

You must retain the *ctlid* returned when creating the control so that it can subsequently be updated. For example:

```
MAP1 CTEXT,S,5,"0"       ! % completed
xcall AUI, AUI_CONTROL, 1, prgbid, ctext, mbst_enable, mbf_progress, "", "",
   status, srow, scol, erow, ecol
```

Initially the progress bar will be empty. To update it, you can use opcode 2 (change), specifying the *ctlid* (`PRGBID` in the above example) returned from the create operation, placing a string representation of the percent completed in the *ctext* parameter, and specifying the *MBST_CHANGE* flag in the *cstate* parameter, e.g.:

```
CTEXT = str(PERCENT) ! percentage complete (round to integer)
xcall AUI, AUI_CONTROL, 2, prgbid, str(pct), mbst_change
```

The above call would move the progress bar over to the position representing whatever percentage is in the variable PERCENT. Note that we omitted unnecessary parameters, which not only saves typing but in the case of ATE would eliminate the need for it to return the status parameter to us, cutting down on network traffic. (In any case you probably don't want to update the progress bar more often than necessary to convey the sense of action. For example, in a report of 100,000 recods, it probably wouldn't make that much sense to update the progress bar for every single record.)

Often a progress bar is displayed inside of a dialog box which contains a message indicating the nature of the activity, an possibly a CANCEL button. As a convenience, we provide a higher-level SBX routine, PROGRS.SBX, which combines these elements (text message, progress bar, optional cancel button, within a dialog box). See the *A-Shell XCALL Reference* for details on PROGRS.SBX.

Tab Control

A Tab control looks something like a series of file folders, where you can see the "tabs" for all of them (unless they scroll out of view) but can see the contents of only one at a time. Another way to conceptualize a Tab control is as a combination of a Groupbox and a row of buttons ("tabs") along the top. Clicking on one of the buttons or tabs sends a keyboard command notification allowing the program to replace the contents of the current groupbox with new contents, depending on which tab was clicked.

The coordinates specified are for the outer rectangle, which includes the space occupied by the tabs. The "client area" of the tab control (the usable area bounded by the tabs at the top and the left, right and bottom edges) acts like a groupbox. Controls placed in this area must specify the ID of the tab control in the *parentid* parameter, and their coordinates are treated as offsets from the upper left corner of this client area. If the number and style of tabs is such that there are multiple rows of tabs, the client area is further reduced by the space taken up the the extra row(s) of tabs, so you should allow extra space if this is a possibility.

As with a groupbox, deleting the tab control deletes all of the controls within it. Perhaps more importantly, using opcode 4 to clear the tab deletes all the controls on the current tab page, a necessary step in switching to a new tab page.

Variations of the basic tab control can be specified by adding one or more of the following to the *winstyle* parameter:

Symbol	Description
TCS_BUTTONS (256)	Tabs appear as buttons and no border is drawn around the display area.
TCS_MULTILINE (512)	If the tabs do not fit on one row, multiple rows are used. Otherwise, a scroll button is placed at the edge of the one row of tabs to allow it to be scrolled horizontally.
TCS_FIXEDWIDTH (1024)	All tabs are made the same width; otherwise the width of each tab is based on the text in that tab.
TCS_FORCELABEL LEFT (32)	Tab labels are left justtified within each tab. This only makes sense with the TCS_FIXEDWIDTH style. Otherwise they are centered.

The tabs are defined by specifying *ctext* in the following format:

CTEXT=Label1~Cmd1~~Label2~Cmd2~~...LabelN~CmdN

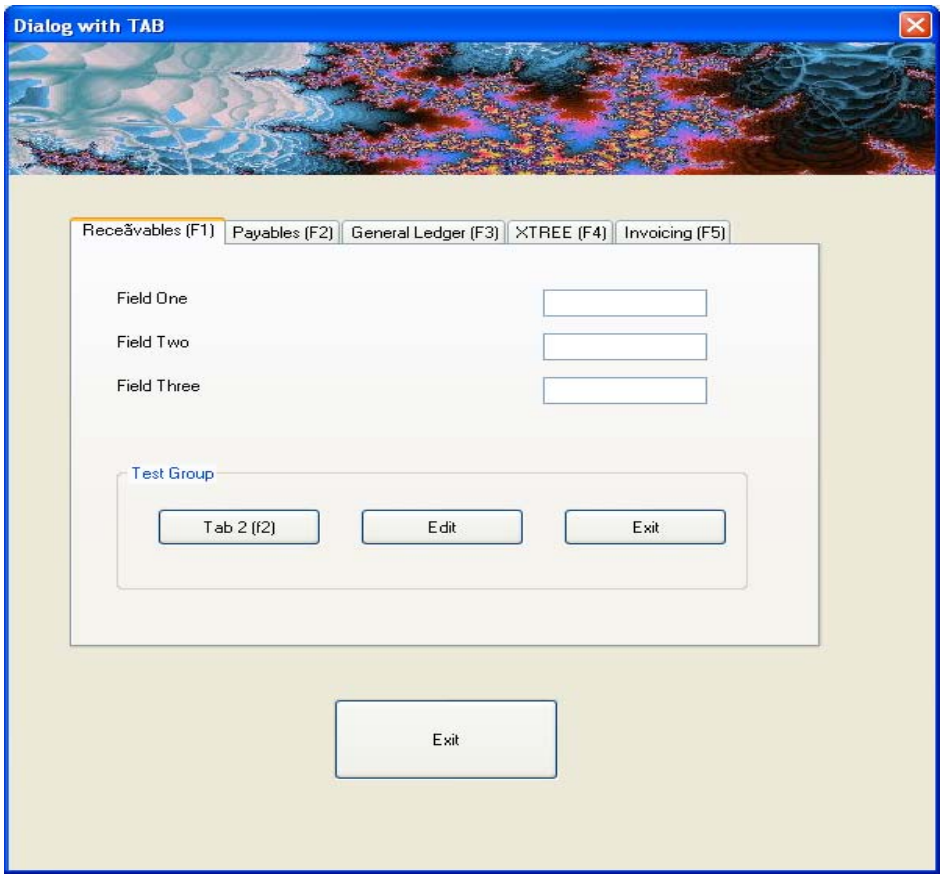
In the above specification, Label1 thru LabelN represent the text strings displayed in each tab, and Cmd1 thru CmdN represent the keyboard command strings each tab sends. As an example:

```
CTEXT="Receivables~" + chr(7) + chr(250) + "101." + "~~Payables~" + chr(7) +
   chr(250) + "102." + "~~General Ledger~" + chr(7) + chr(250) + "103."
```

This would define a set of 3 tabs, "Receivables", "Payables" and "General Ledger". Clicking on the "Receivables" tab would generate exitcode –101 (pseudo function key F101), etc. A program using this arrangement would have to check for these exitcodes (or raw key sequences if your input routine did not provide automatic exitcode conversions), and on receipt, display the appropriate options within the display area of the tab control, according to which tab clicked.

Tab Control Example

See the sample program TABDLG.BP for a good example of the tab control. Here's a screen shot of it, showing a Tab control inside of a modal dialog box:



The colored strip across the top of the dialog is a bitmap file, displayed on the dialog using the AUI "IMAGE" class.

Eventwait

```
xcall AUI, AUI_EVENTWAIT, parentid, ctlid, exitcode {, opflags
      {,timer}
```

The AUI_EVENTWAIT class is really less of an independent "class" than a method relating to the CONTROL class, but is separated out for parameter organization convenience. Its purpose is simply to wait for an event, allowing the user to shift the focus among a range of buttons (plus some other control types) until something is clicked which triggers an event. It could be implemented using an invisible INFLD call in a loop, but this is much simpler to program.

The basic concept is analogous to the dialog manager in Windows or forms manager in VB, which allows the user to move around the form or dialog until some event is triggered that requires the dialog or forms manager to exit (temporarily or permanently) back to the application. In Windows/VB this is done by calling special event handler routines in the application. In A-Shell, this is done by returning from the XCALL with an exitcode and control ID and leaving it to the application to perform whatever action it deems suitable, after which the application can return to the eventwait (or not).

Aside from simplifying the coding of event-driven programs, AUI_EVENTWAIT can often be useful for converting legacy programs which prompt the user to enter a field number to edit, or to choose among a handful of commands that might otherwise be represented as buttons. In the legacy case, the program would be waiting for only one kind of event (the entry of a field number or menu command.) With AUI_EVENTWAIT, the program logic is nearly the same, except that instead of keying in a command, the user can select a field to edit or operation to perform by either clicking on something that generates an exitcode, or by using the arrow and tab/shift-tab keys to move the focus around a collection of buttons or other controls and hitting ENTER to generate the equivalent of a click.

There are several options relating to managing which control gets the focus to start with, which controls can get the focus, and what kinds of events generate exitcodes.

Although AUI_EVENTWAIT will exit on any click event that generates an INFLD-style exitcode, it only allows the focus to be moved (via arrows, tab/shift-tab) to a limited subset of controls that can take the *MBF_TABSTOP* style. Standard pushbuttons automatically get this style, while it can be manually added to checkboxes and radio buttons when they are created. In the case of INFLD controls, there is no way to specify the MBF_TABSTOP style when creating them, but you can pass the *EVW_INFLD* flag to AUI_EVENTWAIT to temporarily confer that style on all INFLD controls in the scope.

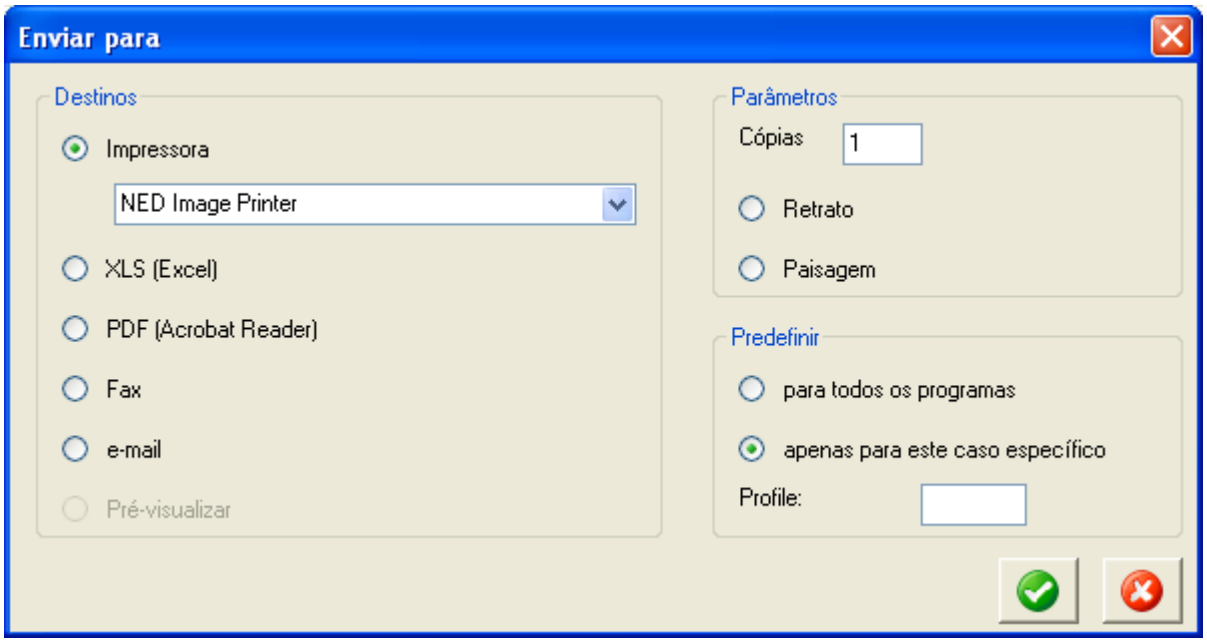
Parameter List

Parameter	Description
parentid	[in] Specifies a parent control to which the button controls of interest are children to. Leave zero if not applicable. See below.
ctlid	[in/out] ID of the control which should start with the focus. If zero, the first control within the group specified by parentid will start with the focus.
exitcode	[out] Returns an INFLD-standard exitcode indicating the event received. Typically this would be the exitcode associated with mouse click on an object, or that of a button that was fired by hitting the ENTER key when it had the focus.(ESC returns exitcode 1, Control-C returns exitcode 10.)

Parameter	Description
opflags	[in] Flags affecting operation (see below).
timer	[in] Optional numeric argument which can be set to the number of milliseconds to wait for an event before the eventwait times out with exitcode 11.

parentid

[in] The **parentid** parameter determines the scope of the controls to which the focus can be moved using the TAB and arrow keys, while waiting for an event. Note that unless the flags *EVW_DESCEND* and/or *EVW_SIBLINGS* are set, the focus will be limited to direct children of the specified group or dialog. To claify this, consider the dialog below:



This dialog is made up of at least 3 groups. (The two buttons at the bottom may be direct children of the dialog, or they may be in their own invisible group, which is common but hard to detect visually.) If we set the **parentid** to the ID of the dialog, and don't set the *EVW_DESCEND* flag, then the user would not be able to move the focus to the radio butons and edit boxes within the groups, because those are all grandchildren (rather than direct children) of the dialog. If the the two icon buttons at the bottom were direct children of the dialog (not part of their own invisible group) then the focus would be limited to those two buttons. (That is, the TAB and arrow keys would not permit the focus to be moved into any of the groupboxes.) The user, could, however, use the mouse to click on anything that generates an exitcode.

Similarly, if the **parentid** was set to the ID of the group "Destinos", then, the user could arrow or TAB around within that groupbox but not go directly to another groupbox without first existing from the event wait operation (unless the *EVW_SIBLINGS* flag was set.).

ctlid

[in/out] ctlid is often set to zero, which causes EVENTWAIT to initially put the focus on the first control within the specified **parentid**. On exit, it will be automatically set to the ID of the control that last had the focus (or was just clicked on). Setting the ctlid to a specific control ID will cause the focus to start there.

If the specified ctlid does not exist, EVENTWAIT returns exitcode 99. Also note that if ctlid does exist, but is not in the group you specified by parentid, then the results might be unpredictable.

opflags

opflags specifies any sensible combination of the symbols below (symbols defined in **Error! Reference source not found.**):

Symbol	Description
EVW_NEXT	Set initial focus on control following the one specified by ctlid. Generally not used, particularly if *ctlid* = 0, since the focus will automatically be placed on the first control within the specified parent group.
EVW_NOWAIT	Set the focus and return (don't wait for an event). This is a bit of an oxymoron, since it directly contradicts the idea of waiting for an event, and thus is only useful in very limited circumstances, where you just want to put the focus on a control while doing some time consuming calculations.
EVW_NOWRAP	Exit from the wait operation with EXITCODE 3 or 5 rather than wrap.
EVW_NOFOCUS	This is another exotic switch, which eliminates the initial step of setting the focus on an appropriate control when starting the eventwait. Probably the only situation where it would be useful is if, due to the context of the program, the application knew that it wanted to start with the focus where it currently was, without knowing where that was..
EVW_NUMERIC	Allow keyboard input.
EVW_DESCEND	Expand scope to include grandchildren, great-grandchildren, etc.
EVW_SIBLINGS	Expand scope to include siblings of parentid.
EVW_INFLD	This flag causes eventwait to treat any EDIT, COMBO, or DATE/TIME control as "hot", meaning that you can move the focus to it using keyboard navigation. Otherwise eventwait ignores controls except for buttons that have the *MBF_TABSTOP* style. (Standard push buttons automatically get the *MBF_TABSTOP* option.)
EVW_PREV	This is the inverse of the EVW_NEXT option, and was provided mainly to satisfy some arcane notion of symmetry.
EVW_SQUELCH	Squelch radio button exits on focus change.
EVW_TABEXIT	This option overrides the normal behavior of the TAB and and Shift-TAB, so that instead of advancing the focus, they simply exit, with exitcode 7 and -35, respectively.
EVW_ACCEL	Allow "accelerator" keys.
EVW_RAW	Raw keyboard input.

Symbol	Description
EVW_HAREXIT	Exit on horizontal arrows. Exitcodes: Left=2, ShiftLeft=-36, Right=12, ShiftRight=-38
EVW_VAREXIT	Exit on vertical arrows. Exicodes: Up=3, ShiftUp = -37; Down = 5; ShiftDown = -39

EVW_NOWRAP

This option controls what happens when keyboard navigation commands (arrows, tab, shift-tab) are used to move the focus around within a group of controls and you hit the "edge". If the flag is not set, then the focus will "wrap" around to the first control in the group. Otherwise, the eventwait operation will exit, setting EXITCODE 3 (up, left) or 5 (down, right) depending on which direction the group was exited. (The *ctlid* parameter will also be set to identify the last focused control, so the distinction between *exitcode* 3 and 5 may not be that important.)

As an example, consider the Tab Control Example dialog above, which features a groupbox containing three buttons. In the default case, without *EVW_NOWRAP*, hitting the left/right arrows, or tab/shift-tab keys, will just move the focus around in a circle between the buttons. The user will have to click on one, or hit ENTER to exit from the eventwait. If the user wanted to edit one of the fields in the tab pane, they could click the "Edit" button. As an alternative, you could use the *EVW_NOWRAP* flag, in which case, merely using the arrow keys would trigger an exit from the eventwait, allowing the program to refocus on one of the edit fields. (See EVW_SIBLINGS and EVW_DESCEND flags for other approaches to allowing the user to navigate around the dialog.)

EVW_NUMERIC

This switch activates a feature that would probably only make sense to programmers/users most comfortable with the kind of screen whose fields are all numbered, and where you the main loop consists of prompting the user for a field number to edit. When migrating such a program into the GUI environment, you might replace the "Enter Field #" prompt with an eventwait operation, which allowed the user to click on a field to edit, or use the arrow keys, or just enter the field number and hit ENTER. In the latter case, the eventwait will exit with *ctlid* set to the number entered, and *exitcode* set to 0. See EVW_RAW for a similar option and a warning note.

EVW_DESCEND

This flag allows the focus to be moved (via the keyboard arrows and tab/shift-tab) into child subgroups of the *parentid*. Otherwise keyboard navigation is limited to the immediate group. For example, in the sample dialog above (see parentid) there are three or four separate groups in the dialog. If you start an eventwait with parentid set to one of them but the *EVW_DESCEND* flag not set, the user will be able to navigate around the controls in that group, but not leave the group (except by clicking or some other event that triggers an exitcode). This might be useful if you wanted to do some validation logic on the group as a whole before moving to another group. On the other hand, if you wanted to allow the user to navigate around all of the groups without requiring program intervention, then you could set the parentid to the ID of the dialog itself and set the *EVW_DESCEND* flag.

EVW_SIBLINGS

This flag is similar in concept to the *EVW_DESCEND*, except that it extends the scope of controls which can be navigated to via keyboard navigation commands to groups and controls that have a sibling relationship to the *parentid*. The effect is essentially the same as if instead of setting *parentid* to a particular group, you set it to the parent of that group (or the dialog ID), and set the *EVW_DESCEND* flag.

EVW_SQUELCH

This oddly named feature, like its namesake in the world of radios, turns down "the noise" when navigating among a collection of radio buttons. With buttons and checkboxes, the focus can be moved from one control to the next without actually changing the selection. But with radio buttons, the act of changing the focus with the arrow keys also changes the selection, and thus is essentially the same as clicking on a button. If each button has an exitcode associated with it, then each time you hit the arrow key to move to the next radio button, the eventwait will exit. This might be appropriate when, depending on which radio button is selected, you might need to enable/disable some other controls in the dialog. For example in the output dialog above (see parentid), if you select the printer option, then you would want to enable the combo box containing the list of printers, and set the focus on it. But if you selected, say, PDF output, then the printer selection combo box can be disabled, along with the Copies option. The immediate exitcode as you move the focus among the radio buttons allows the program to perform these actions.

But if you don't need to perform any immediate action just because the radio button selection was changed by arrowing through the group, then use the EVW_SQUELCH option, and wait for some more important event to terminate the eventwait, after which you can go back and query the buttons.

To advance the focus out of the current radio button group, without changing the currently selected radio button, use TAB (or Shift-TAB).

EVW_ACCEL

This flag allows alphanumeric keystrokes to act as "accelerator" keys, automatically selecting and clicking on the matching control, if present. You can define an accelerator key association for a control by inserting a "&" in front of the desired character in the control's text. For example, a button defined using "E&xit" will use "x" as the accelerator key. (The & will not appear, but the subsequent character will be underlined.) In this case, hitting "x" will have the same effect as clicking on the button.

EVW_RAW

Causes nearly all keystrokes to be returned "raw" by setting the *ctlid* to the ASCII value of the keystroke and *exitcode* to 0. The only exceptions are the TAB, Shift-TAB, and arrow keys, which retain their navigation function. This is similar to *EVW_NUMERIC*, in that the *ctlid* parameter is used to return a user-input code which has nothing to do with an actual control ID.

Since the returned *ctlid* may not have anything to do with a valid control ID, take care not to pass the value back as input to a subsequent EVENTWAIT operation (or else it will probably abort with *exitcode* 99, meaning invalid control ID.)

Comments

When converting from a traditional procedural program to an event-driven one, you will probably encounter situations where used to have an input prompt to wait for the user to select among many choices or confirm/cancel an operation. In the GUI model, such an input prompt may seem silly, since the layout of the screen (buttons, other things to click on) makes it obvious what the user needs to do without being prompted to type something. This EVENTWAIT class is perfect for such situations. Examples include:

- Menus, where in the text version you might have a prompt like "Enter Selection:" but now you have buttons.
- At the bottom of an input form, where in the text version you might have had an "Any Change?" prompt, but now you have buttons such as "OK" and "Cancel", with the implicit understanding that the user can just click on a field to edit it.
- Dialogs, especially those containing checkboxes and radio buttons (which do not require any special procedural coding and can instead just be queried upon exiting the dialog).

Although you don't need to specify a parent group, this concept is generally much clearer if you do group the relevant buttons within a parent. The best way to do this is with a Groupbox Control. Note that the group box need not take up any extra space or even be visible. (You can set its initial state to *MBST_HIDE* and its size to one row high, and place the buttons right on top of it.)

The BTNMNU.SBX routine encapsulates this logic in a wrapper making it easy to create and use (i.e. wait on) groups of buttons.

The TSTEVW sample program (in the A-Shell Open Source Library) provides detailed examples of using EVENTWAIT, both via BTNMNU.SBX and via XCALL AUI.

EVENTWAIT only allows the focus to be moved amongst controls that have the *MBF_TABSTOP* property. This property is automatic for regular buttons, and may be applied manually to check boxes and radio buttons. No other control type currently supports the *MBF_TABSTOP* property, although the *EVW_INFLD* flag will cause EVENTWAIT to act as if all INFLD controls had the *MBF_TABSTOP* property.

Environment

```
xcall AUI, AUI_ENVIRONMENT, opcode, guiflags
```

The ENVIRONMENT class keeps track of properties and capabilities of the interface environment that affect the kind of user interface a program should use. It currently supports just two methods, get (0) and set (1) GUI flags. (As a practical matter you would probably never set any of these properties yourself. They are set automatically by A-Shell when a session is launched.)

This is equivalent to xcall *MIAMEX, 129, opcode, guiflags.* Values for *guiflags* are shown in the table below, and are defined in **Error! Reference source not found.**.

Symbol	Meaning
AGF_LWG	Local Windows w/ GUI (e.g. PCTDVG)
AGF_LWN	Local Windows w/o GUI (e.g. PCTDV)

AGF_ATE	Running ATE on client
AGF_RWN	Remote windows (ATS)
AGF_TNT	Telnet
AGF_ASH	A-Shell
AGF_THEMES	XP Themes are active
AGF_LOCWIN	Local Windows (AGF_LWN or AGF_LWG)
AGF_ANYWIN	Any windows platform (_LWN or _LWG or _RWN)
AGF_GUIEXT	GUI extensions avail (AGF_LWG or AGF_ATE)

Here are some examples of using this information. All of them would start with retrieving the current *guiflags* as follows:

```
xcall AUI, AUI_ENVIRONMENT, 0,GUIFLAGS       ! retrieve GUI flags
```

Before creating a graphical user interface with components from the AUI CONTROL class, you might want to check for either *AGF_LWG* or *AGF_ATE*. Since these are combined in *AGF_GUIEXT*, we can simply check it as follows:

```
if (GUIFLAGS and AGF_GUIEXT) then ?TAB(-10,x);.....
```

If you wanted to display a file for the user, either using NOTEPAD (if applicable) or else EZTYP (if not), you might use the following logic:

```
if (GUIFLAGS and AGF_LOCWIN) then &
   xcall HOSTEX,"NOTEPAD "+F$ &
else if (((GUIFLAGS and AGF_ATE)#0) and &
    ((GUIFLAGS and AGF_TNT)#0)) then &
    call XFER'TO'PC :&
    call LAUNCH'NOTEPAD'ON'CLIENT &
else &
    xcall EZTYP,F$
```

In the above example, if running Windows locally, we can just launch NOTEPAD using XCALL HOSTEX. Otherwise, if running ATE on the client over telnet, we could transfer the file to the PC and then launch NOTEPAD, both of which could be done via TAB(-10,x) commands. Otherwise, we could just use EZTYP on the server. (You might also want to test for other telnet emulators, such as ZTERM, which are capable of doing file transfers and launching commands on the client, but that exceeds the scope of this example.) Note that when testing multiple AND conditions, you need the extra set of parentheses and the #0 test as shown above to get the desired effect. Otherwise the confusion between the logical and arithmetic AND in BASIC will cause the test to fail.

If you merely wanted to test whether you were running under A-Shell Windows or A-Shell/UNIX (or some other platform, presumably AMOS), you could do this:

```
if (GUIFLAGS and AGF_ASH) then       ! running A-Shell
    if (GUIFLAGS and AGF_ANYWIN)     ! any Windows platform
       ? "A-Shell Windows"
    else
       ? "A-Shell UNIX"
    endif
else
    ? "Non A-Shell (AMOS?)"
endif
```

Menu

```
xcall AUI, AUI_MENU, opcode, menuid, mnutxt, mstate, mtype, cmd,
      func, mstatus

xcall AUI, AUI_MENU, opcode, mdfspec, mstate, mstatus
```

The MENU class deals with the A-Shell menu bar, offering methods for adding and deleting menu items. The first form of the syntax above is the original method, which is still the most flexible, but most difficult to use. The second form vastly reduces the amount of code you need to write. We will start with the original version since its concepts are used in the simplified version.

Traditional Method

opcode (numeric) should be one of the following:

Value	Meaning
0	No change to the menus; just check to see if a menu item exists
1	Add new menu items
2	Change state of existing menu items
3	Delete menu items

menuid (any type) specifies the top level menu number or its name, according to the table below:

Value	Meaning
0	Top level menu as a whole. Use this when adding or deleting an item that appears on the horizontal menu bar (which starts with items “File”, “Edit”, “Settings”, “Help”.
1	The “File” menu. Use this to add or delete items to the drop down menu that appears when you click on “File”.
2	The “Edit” menu. Use this to add or delete items to the drop down menu that appears when you click on “Edit”.
3	The “Settings” menu. Use this to add or delete items to the drop down menu that appears when you click on “Settings”.
4	The “Help” menu. Use this to add or delete items to the drop down menu that appears when you click on “Help”.
5 or <name>	This would be the number of the first item added to the top level menu bar. You could also refer to it by the same text that you specified as *mnutxt* when you added it. Use this to add or delete items to the drop down menu that appears when you click on this menu.
6 – 14 or <name>	Same idea as for five, but for the sixth through fourteenth items on the top menu bar.

mnutxt (string, 31 char max) specifies the text that will appear in the menu item. You may precede one letter with “&” to allow that letter to be used with the ALT key as a hot key for the menu. For example, “&Custom” would display as “Custom” and could be selected via ALT-C. *mnutxt* also serves as the identifier for the menu item, allowing it to be selected for later deletion, so it should be a unique string.

mstate (numeric) indicates the state of the menu item:

Value	Meaning
0	Enabled
1	Disable (grayed out)

mtype (numeric) specifies any valid combination of options from the table below:

Value	Meaning
MBF_CMD LIN	Normal. Selecting the menu item causes the contents of *cmd* to be executed as a Windows command line, as if it had been executed via XCALL HOSTEX. As with XCALL HOSTEX, you can refer to the current A-Shell executable (and its current MIAME.INI file) via the macro symbol $ASHELL, and you can use the suffix characters available to HOSTEX. For example, "$ASHELL –e run armenu" would launch a new A-Shell instance, run the program armenu, and suspend the current session until armenu exited.
MBF_DLL	DLL. Selecting the menu item causes the function (*func*) with the associated DLL (defined in *cmd*) to be loaded and executed.
MBF_SUB MNU	Submenu. Selecting the menu item will cause a submenu to appear. Currently only one level submenu is supported (i.e. all submenus must belong to the top horizontal menu bar).
MBF_SEP	Separator. Item displays a s horizontal separator bar. Separators cannot be selected and only serve to visually organize submenus. Even though separators do not display the *mnutxt* contents, they still need a unique *mnutxt* string to allow the item to be deleted later.
MBF_KBD	Keyboard. Selecting the menu item causes the contents of *cmd* to be forced into the keyboard buffer. This technique may minimize the impact on the program, since to it, menu selections may appear just like ordinary keyboard responses, but you have to be careful to disable such menus when the program context is not correct to interpret such responses. See *cmd* (below) for further notes on designing and encoding keyboard sequences.
MBF_SHL EXC	Shell execute. Selecting the menu item causes the contents of *cmd* to be interpreted as an object (file or URL) to be opened using the associated application defined in the Windows registry. For example, if CMD="http://www.microsabio.com", then selecting it would launch the browser and go to the specified web page. If CMD= "mydata.xls", selecting the menu would probably launch Excel to open the spreadsheet. If the file object specified in CMD does not contain a drive or directory specifier, then it will be assumed to be in A-Shell's "DOC" subdirectory (e.g. C:\VM\MIAME\DOC). Otherwise, you need to specify a native pathspec (unless the file is in the system search PATH). Refer to MIAMEX 3: Perform FSPEC on AMOS string for converting AMOS directories to their native equivalents.

cmd (string) should be set to the command, DLL, keystrokes, or object, according to the *type*. It is ignored for the submenu (16) and separator (2048) menu item types.

The most convenient approach for handling menu options as keyboard commands is often to make them emulate INFLD function keys (thus allowing the menu selections to interrupt existing input operations while still allowing your code to respond differently depending on the context.) To emulate a function key, set CMD to a string consisting of a Control-G (ASCII 7) followed by "1" thru "9" for F1 thru F9, or "A" thru "Z" for F10 thru F35. ASCII characters higher than "Z" will result in correspondingly higher function key values.

Encoding control characters in the *cmd* as described above is somewhat awkward at best and makes it impossible to store them in a parameter file, as is the case with the simplified method (described below). As an alternative, you can also encode them using the ^X notation, (where ^X is interpreted as Control-X, for any character A-Z or [,], \, underline, or another caret). Thus the *cmd* string "^GA" would be treated as Control-G followed by the letter A, and "Hello^M" would be treated as the word "Hello" followed by a carriage return (Control-M). *func* (string) must contain the name of the function to execute within the DLL whose name was specified in *cmd*. It will be ignored except when the *type* is 1.

mstatus (F,6) returns a code indicating the result of the operation:

Value	Meaning
0	OK, or for opcode 2 indicates item was previously enabled
1	OK, or for opcode 2 indicates item was previously disabled
-1	Add or delete menu function failed
-2	Menu (*mnutxt*) not found during change or delete operation
-3	Out of memory (unable to allocate menu storage buffers)
-4	Exceeded maximum number of added menus (currently 125)
-5	No menu buffer allocated
-6	Illegal opcode
-7	Menu (*mnutxt*) already exists

Simplified Method

The simplified menu calling syntax uses just five parameters, as shown here:

```
xcall AUI, AUI_MENU, opcode, mdfspec, state, status
```

opcode, *state*, and *status* have the same meaning and usage as for the traditional method, described above.

mdfspec (string) should contain the filespec (AMOS or native format) of a "Menu Definition File" which consists of lines of the following format:

```
;(Blank lines and lines starting with semicolon are ignored)
;
MENUID,MNUTXT,TYPE,CMD
MENUID,MNUTXT,TYPE,CMD
etc.
```

By convention, the normal extension for such a file is MDF (Menu Definition File). If the *mdfspec* parameter is "", then it is assumed to refer to A-Shell's help menu definition file, $MIAME\doc\ashelp.mdf.

If running on a UNIX server with an ATE client, the mdfspec will be resolved on the PC side and must reside there in advance of the call. (You can transfer it via FTP necessary.) If just a filename and optional extension given, it will be treated as being in the local ATE current ppn, which by default is DSK0:[1,4].

The file can specify any number of menu items (subject to the maximum of 125 custom menu items at any one time), all of which will be added, deleted, or enabled/disabled in one operation.

Each of fields *menuid*, *mnutxt*, *type* and *cmd* have the same meaning as in the traditional calling format. As an added convenience, the TYPE field may also be specified as a three-character mnemonic:

Type	Mnemonic
0	CMD
1	<not supported in MDF format>
16	SUB
2048	SEP
65536	KBD
131072	REG

menuid can also be specified as a mnemonic, as shown in the table below. Note that this is the same table with the same values as shown and as used in the "Traditional Method." In the "Simplified Method," however, the mnemonics are allowed, whereas in the "Traditional Method" they are not.

MenuID	Meaning
0 or "Top"	Top level menu as a whole. Use this when adding or deleting an item that appears on the horizontal menu bar (which starts with items "File", "Edit", "Settings", "Help".
1 or "File"	The "File" menu. Use this to add or delete items to the drop down menu that appears when you click on "File".
2 or "Edit"	The "Edit" menu. Use this to add or delete items to the drop down menu that appears when you click on "Edit".
3 or "Settings"	The "Settings" menu. Use this to add or delete items to the drop down menu that appears when you click on "Settings".
4 or "Help"	The "Help" menu. Use this to add or delete items to the drop down menu that appears when you click on "Help".
5 or <name>	This would be the number of the first item added to the top level menu bar. You could also refer to it by the same text that you specified as *mnutxt* when you added it. Use this to add or delete items to the drop down menu that appears when you click on this menu.
6 – 14 or <name>	Same idea as for five, but for the sixth through fourteenth items on the top menu bar.

Any of the fields in the MDF may optionally be enclosed in quotes – this in only mandatory when the field itself contains a comma.

> See the notes in the TYPE table and under the CMD definition for the traditional method above for details about how to format the CMD string for the various TYPEs.

As an example, consider the following XCALL and sample MDF file. The comments in the sample.mdf should hopefully make clear what it does.

```
xcall MIAMEX, MX'WINMMU, 1, "sample.mdf", 0, STATUS
```

Sample.mdf

```
;Add a separator bar to File menu:
FILE,"fs1",SEP
;Add item to "Select Source" (implemented my launching another
;  instance of A-Shell)
; (Note -e for auto exit)
FILE,"Select Source...",CMD,"$ASHELL -e run selsrc"

;Next, add a new top level menu called "Tools"
TOP,"&Tools",SUB

;Add a calculator and a website link to it...
"&Tools","Calculator",CMD,"calc.exe"
"&Tools","Web Search",REG,"http://www.google.com"

;Add a separator and two PDF document files to Help menu
HELP,"fs2",SEP
HELP,"User Guide",REG,"UserGuide.pdf"
HELP,"Troubleshooting Reference",REG,"TroubleRef.pdf"
```

If an error occurs, the operation will abort at that point, which should hopefully help you pinpoint the cause.

It is usually necessary to delete menus in the reverse order from the way they were added (in order to avoid the confusion caused by later menus changing position due to the deletion of earlier ones). The simplified method takes care of this difficulty for you, but only within a single MDF file. If you use multiple MDF files and want to be able to delete them independently of each other, you should having more than one MDF file add menu items to the same top level submenu, and you'll need to refer to the top level submenus by name rather than number.

Refer to the sample program ASMENU.BAS which is included in the A-Shell release (in [7,376]), and to the Application Developers Guide for more details.

Image

```
xcall AUI, AUI_IMAGE, opcode, handle, status {,params...}
```

Except for two issues, AUI Image is identical to the IMAGE subroutine, which see for full information. The differences are:

- The syntax for AUI Image is as shown above; "xcall AUI, AUI_IMAGE" here replaces "xcall IMAGE" in **IMAGE.SBR**.

- AUI Image supports only opcodes one (load), two (close), three (display), four (open and display) and six (remove).

Window

```
xcall AUI,AUI_WINDOW, flg {,lft, top, rgt, btm {,rows, cols {,tsts,
      bsts}}}
```

The WINDOW class is used for querying and changing the parameters of the window, such as its size, position, display state, number of rows and columns, etc.

Parameters

FLG (numeric) specifies the window state desired, or the option to query the current state:

Value	Meaning
-1	Query the current settings and update all of the passed parameters accordingly.
0	Hide the window
1	Display the window in its normal size
3	Maximize the window
6	Minimize the window
9	Restore the window (from minimized or maximized) to the normal size
+64	Causes the window to initially be reset based on the current settings file (as edited on the Settings menu and saved with the File..Save menu).

To just restore the window to its initial state, as specified in the settings file, set *FLG*=65 and either omit all the other parameters or set them to 0 (-1 for *TSTS* and *BSTS*).

lft, top, rgt, btm (numeric, optional) specify or retrieve the coordinates of the Window in standardized units that range from 0,0 for the upper left corner of the screen to 10000,10000 for the bottom right. You may leave these 0 to retain the current window coordinates (if you just want to change the ROWS/COLS or status lines.)

rows ,cols (numeric, optional) specify or retrieve the number of rows and columns which the window is logically divided into. You may leave these 0 to retain the current window coordinates (if you just want to change the coordinates or status lines.)

tsts, bsts (numeric, optional) specify or retrieve the visible / invisible state of the top and bottom status lines (1=visible, 0=invisible). You may leave these -1 (or omit them) to retain the current status line state.

See the sample program AUIWIN for an example of using AUI, "WINDOW" to change and later restore the window parameters.

HTMLHelp

```
xcall AUI,AUI_HTMLHELP, op, status, hfile {,htopic{, row, col}}
```

The HTMLHELP class provides means of displaying context-sensitive help using Windows CHM/HTML help system. By establishing links between topics in your help file and operations in your program, you can call the appropriate topic when the user invokes help. As currently implemented, the specified CHM file opens and displays the referenced topic.

Parameters

OP (numeric):

Symbol	Meaning
HHOP_ DFLT	Set default application help file. This allows help topics to be specified to INFLD without having to specify the help file each time.
HHOP_ DSPID	Display topic based on HFILE (or default), and a numeric topic ID in HTOPIC. (You'll have to consult the creator of your CHM files to get a list of the numeric topic ID numbers.) If HTOPIC is 0, the help file is launched in its default state.
HHOP_ DSPSTR	Same as OP 1 except HTOPIC should be a string which specifies the internal HTM name for the desired topic. These internal names are typically of the form "/topicname.htm", e.g. "/comio.htm". The leading slash is typically necessary. You might be able to determine these names by using a binary DUMP utility on the CHM file (such as DUMP.LIT). As with OP 1, if HTOPIC is "", the help file is launched in its default state.
HHOP_ POPUP	Display a pop-up message. The text of the message (up to 2048 chars) should be specified in HTOPIC, and HFILE should be "". If ROW and COL are specified and non-zero, then the top center of the pop-up box will be positioned at that point. Otherwise, the pop-up box will be positioned just below the current cursor location.

status

If you want a return status, specify an F,6 variable, in which case, >=0 indicates success, and <0 indicates failure. The most likely failures would be that the CHM system isn't supported (due to running on UNIX without ATE or perhaps an old version of Windows), or the help file/topic could not be located. If you don't care about the return status, specify this parameter as "". (This eliminates the need for the ATE client to return anything, which is a minor optimization.)

HFILE (string)

File filespec of CHM file. May be in AMOS or native format and may contain environment variables using the %ENV% syntax. Note that you may put your help files in the same directory as the A-Shell help files, in which case you can reference them by:

```
HFILE = "%MIAME%\doc\filename.chm"
```

(If the environment variable %MIAME% is not explicitly defined, it will be interpreted as pointing to the directory where the miame.ini is.)

HTOPIC (must be string for OP 2 & 3, may be numeric for OP 1)

Specifies the topic name (<topic>.htm) (OP 2) or ID (OP 1) or the actual text to display (OP 3). Text may contain embedded CRLF characters for line breaks.

ROW,COL (numeric, optional)

These apply only to OP 3, and only if you want to position the help window somewhere other than just below the cursor location. (For large messages, you might want to position it at something like ROW=2,COL=40.)

This function works in A-Shell/Windows and ATE. It may also be called via MIAMEX,137 (replacing "HTMLHELP" with 137).

Other Control Topics

ATE Optimization

In the ATE environment, any operation that requires a return status or other response from the client workstation introduces a small delay. The delay is not significant when doing just a few operations, but it can add up when large numbers of operations are performed in sequence. The simplest way to minimize the problem is to not ask for a returned status (see cstatus) or *ctlid* when creating controls (assuming that you don't need to use the *ctlid* later.) But when you do need a return status or control ID, then an alternative way to speed up the communication with the client is to create a batch. See Batch Operations.

XP Themes

Windows XP includes a technology the goes by the name of "Visual Themes", or "Visual Styles", or just "Themes", which essentially takes control of the appearance of most interface objects to give them a fancier and more integrated look. The most obvious way to recognize when Themes are active is to move the mouse over a button to see if the button changes its appearance (depending on the particular Theme, adding a highlight around the rim, possibly changing the color.) You will also notice that most controls have more rounded edges and often they are filled with a gradiant rather than a uniform color. The standard XP Theme gray background color is also quite a bit lighter than the gray used in Windows 2000 or W9x. Most people agree that Themes provide a more aethetically pleasing user interface look, and even if if people don't recognize the specifics, they can generally recognize instantly the difference between an "XP-Style" application and a "Classic Windows" application, with the latter being almost considered derogatory these days.

There is, however, a downside to Themes, and that is that by taking control of the rendering of interface objects, we lose the ability to assign arbitrary colors to things. Although there are some exceptions, in order to maintain a uniform look, A-Shell ignores your color specifications when Themes are active. The primary exception relates to the foreground color of static text objects. If you want to override the color in all cases, including when Themes are active, then you must add 64 to the color palette number. For example, if 3 is magenta, specifying 3 for the foreground color of a text control will result in magenta in a non-Themed environment, but will revert to the Theme-specified color when Themes are active. To override this and force magenta in all cases, add 64 (+3 = 67).

As of build 903, A-Shell/Windows and ATE support XP Themes through the use of a "manifest" file (ashw32.exe.manifest) which is installed in the same directory as the ashw32.exe executable. If the manifest is not present, then Themes will not be activated. Even if the manifest is present, Themes will not be activated for pre-XP versions of Windows. The best way to disable Themes for A-Shell under XP is to click the "Disable Visual Themes" checkbox on the Compatibility tab of the Properties dialog of the shortcut that launches A-Shell.

There is a known bug in XP Themes relating to the Tab Control: the gradiant fill of the tab body supports a maximum height of 600 pixels. If your tab body is higher than that, the portion beyond 600 pixels will be white instead of the gradiant gray. (It's a minor visual anomaly, will probably be fixed in an XP Service Pack, and in most cases can be avoided by not making the tab controls too large.)

B64ENC

```
xcall B64ENC, ifile, och, status
```

B64ENC.SBR converts the file whose AMOS or native filespec is in the *ifile* parameter to Base 64 encoding format, writing the result to the open file channel *och*. (*och* must specify the channel of a file previously opened for output.) On return, *status* (F,6) will be set to the number of bytes written, or <=0 for errors. This subroutine is used internally by **EMAILX.Error! Reference source not found.** to convert email attachments to MIME format, and would be mainly of use to programmers developing their own email utilities.

BASORT

Random file syntax:

```
xcall BASORT, ch, reccnt, recsiz, k1siz, k1pos, k1ord, k2siz, k2pos,
     k2ord, k3siz, k3pos, k3ord, k1typ, k2typ, k3typ
```

Sequential file syntax:

```
xcall BASORT, chin, chout, recsiz, k1siz, k1pos, k1ord, k2pos,
     k2siz, k2ord, k3siz, k3pos, k3ord
```

A-Shell's implementation of **BASORT.SBR** is fully compatible with that under AMOS, optimized to make the best use of memory available. The sorting process is based around the quicksort algorithm, but up to three different types of sort may be performed depending on the amount of free memory. In the default case, the free memory is taken from the A-Shell partition. However, under systems with large amounts of efficient virtual memory available (e.g. UNIX and Windows), you may specify the option SBR=MALLOCSORT to MIAME.INI to cause A-Shell to dynamically allocate, if possible, a chunk of memory large enough to sort the entire file in one quicksort operation. See the *A-Shell Setup Guide* for more details on the **SBR** options.

The parameters are all numeric (any type). The key type values (*k1typ, k2typ, k3typ*) are 0 for string (default), 1 for floating point, and 2 for binary.

In the case of the sequential file sort, there is no option for key types because the keys must be string type. Also, since the record lengths can be variable, you should specify a Recsiz as large or larger than the largest line in the file (up to a maximum of 4096). The file to be sorted must be open for input on ChIn, and the program must supply the channel (*chout*) of an output file that you want the file sorted into. If you want to sort the file on itself, you may specify *chout* as the same value as *chin* (in which case it only needs to be open once for input). If you want to re-read the sorted output file, you will need to close it and reopen it for input.

Here is a description of the three sort algorithms:

SMALL sort

The entire file is loaded, quicksorted and then saved to disk. This will occur if there is sufficient memory to load the entire file. It is extremely fast.

MEDIUM sort

The keys are extracted from the file, and then quicksorted. A new file is then created by reading the records from the old file in the order of the quicksorted keys. This will occur if there is sufficient memory to load a tag file consisting of the sort keys and record numbers. It is reasonably fast.

LARGE sort

The file is quicksorted in chunks into two separate files. A disk-based poly-phase merge sort is then used to repeatedly merge these until a single sequence of sorted records results. This type of sort occurs in all other situations. It is quite slow.

The type of sort being used may be determined by specifying:

```
TRACE=AMSORT
```

in the MIAME configuration file, MIAME.INI (see the *A-Shell Setup Guide*) or with the command:

```
SET TRACE AMSORT
```

from the dot prompt.

Among other uses, the tracing display may be useful for analyzing the performance of a LARGE model sort (since the individual phases are shown).

Comments

As with **BASORT** under AMOS, records with the same sort key are guaranteed not to be re-ordered in the resulting file.

The normal limit on memory allocation with the MALLOCSORT option is 8MB, but you can change this with the MALLOCLIMIT=<bytes>{K}{M} parameter in the MIAME.INI file. Bumping this up, to say, 32MB, on a typical modern desktop PC can make a huge difference when sorting files larger than 8MB (but smaller than the limit you specify).

If you're looking for a more powerful sort routine (i.e. faster, able to handle more keys, able to perform data massaging beyond mere sorting, etc.) we have licensed the OptTech Sort routine and have customized it to understand most of our data structures. Aside from offering additional keys and functionality, it is considerably faster at sorting large files than BASORT. It can also perform fancy data manipulation tricks, like removing duplicate records, adding summary records, creating tag files, etc. But, the calling interface is different, it doesn't handle certain file types, and there is a licensing fee. If interested, please contact us for more details.

Also see SORTIT for sorting arrays in memory.

BITOPS

```
xcall BITOPS, opcode, chfld, pos, result, result2
```

BITOPS.SBR was developed to facilitate the use of large bit fields (bitmaps). These are quite useful when you want to keep a single binary flag for each of many items. The bit field can be held in memory in a variable of type X, or in a random file.

Parameters

opcode (any numeric type) specifies the operation:

Value	Meaning
1	Clear bit #<*pos*>; if <*pos*> = -1 or is omitted, clear all bits in bitmap
2	Set bit #<*pos*>; if <*pos*> = -1 or is omitted, set all bits in bitmap
3	Counts all bits in bitmap; return number of 0s in *result*, number of 1s in *result2*
4	Set *result* to bit # of first 0 bit on or after given *pos*. If no more 0 bits, return –1. At same time, set *result2* to bit # of first 1 bit on or after *pos*.

chfld
is either the bit field (if type X) of any size (first bit is considered bit #0); if any type other than X, *chfld* is interpreted as an open random file channel (containing the bit field).

pos (any numeric type)
is used to specify the bit # we are operating on or starting from (see opcode table above).

result, result2 (any numeric types)
are used to return the results (see opcode table above).

BLOCKS

```
xcall BLOCKS, device, blks, cblks, {,tblks}
```

BLOCKS.SBR allows a program to check how much disk space is free on the specified device. The device must be specified *without* the trailing colon (e.g. DSK0 or EXT1) and must be a device defined in the MIAME.INI file. The number of free blocks (512 bytes per block) on the device is returned in both the *blks* and *cblks* (contiguous blocks) parameters, which must be 6 byte floating point variables.

There are a couple of idiosyncrasies to note about this routine. The first is that under A-Shell, there is no particular significance to *contiguous* blocks, since there is no particular significance to them under Windows or UNIX. This is why it is always equal to the number of free blocks. You do not need contiguous blocks to allocate random files like you do under AMOS. The second is that 512 byte AMOS-style blocks have no particular significance either to the host operating system. Under Windows and UNIX, the minimum allocation unit is related to something other than physical disk blocks (i.e. clusters or inodes) and for that matter, the physical disk block size may or may not be 512. Finally, since A-Shell does not impose any kind of partitioning on the pseudo-AMOS devices, it is really the free space on the logical host disk that is being measured. For example, `DSK0` may be mapped to `/vm/miame/dsk0` and `DSK1` to `/vm/miame/dsk1`, but both of those are part of the same UNIX file system. So they will both show the same amount of free space. (Do not get fooled into thinking that that much free space is separately available on both devices, since it is really only one device from the host operating system's point of view.) Despite all of these quirks, for AMOS applications, it is still a useful approximation to the amount of free space available.

Note that if the device cannot be accessed, the block count will be returned as –1.

The optional last argument, *tblks* (f,6), will return the total number of blocks (both free and unused) on the device.

CCON, CCOFF

```
xcall CCON {,flag}

xcall CCOFF
```

CCON.SBR enables or disables Control-C. When called with no arguments, or if the argument passed has a non-zero numeric value, then it enables Control-C. When called with an argument that evaluates to zero, it disables Control-C and is therefore equivalent to **CCOFF.SBR**.

CCOFF.SBR disables Control-C. This might be useful in a program at the start of a critical update sequence that you did not want the operator to be able to interrupt. It is also equivalent to SET NOCTRLC (from the dot prompt).

CGIUTL

```
xcall CGIUTL, opcode {,param1 {,param2 ...}}
```

CGIUTL.SBR contains several utility functions which are very handy when writing CGI programs. The calling format and parameters beyond opcode depend on the value of *opcode*, so we will consider each case separately, with the value of OPCODE inserted into the syntax examples below. *Opcode* itself can be any numeric format. See the *A-Shell Development Guide* for more details on using **CGIUTL.SBR**.

Opcode 0

```
xcall CGIUTL, 0, status
```

opcode 0 simply checks if A-Shell was launched with the –cgi switch, returning 1 (in *status*) if so, else 0. This can be useful when writing programs that work differently depending on whether they are running inactively or under control of a web server.

Opcode 1

```
xcall CGIUTL, 1, string, status
```

opcode 1 retrieves the entire *stdin* into the *string* parameter. (The web server will have put all of the form information to be passed to the CGI program into *stdin*, provided that you set the form method to "POST.") Returns *status*=0 for ok, 1 for overflow (*string* too small), or -1 if not in CGI mode (i.e. –cgi switch not specified). This is considered a utility operation and is probably not necessary since most of the other operations are capable of reading the *stdin* directly.

Opcode 2

```
xcall CGIUTL, 2, parmname, parmval, status {, string}
```

opcode 2 retrieves the web form parameter whose name is specified by *parmname*, into the string variable *parmval*. Returns *status*=0 for OK, or -1 if the parameter is not found, or –2 if not in CGI mode. If *string* is specified it is used in place of *stdin*. (If you previously used *opcode* 1, then you must specify the *string* returned in that operation as input into this one.) This is the way your program can find out what the user keyed into the various fields on the web form.

The form which invokes the A-Shell instance, which in turn uses **CGIUTL**, must use form action "POST" (not "GET").

Opcode 3

```
xcall CGIUTL, 3, file, status {varsub1 {,varsub2 ... {,varsub27}}}
```

opcode 3 copies the disk file (whose name is specified by *file*) to *stdout,* making variable substitutions as it goes. This is useful for building a new web page from a template file with dynamic values inserted in place of static variables. The optional parameters *varsub1* through *varsub27* may each contain one or more substitutions of the form "NAME=VALUE", to cause any occurrence of NAME within the file to be replaced by VALUE. To allow for more than 27 substitutions, each *varsub#* parameter may contain multiple substitutions, separated by a "+" character. (To include a "+" or a "="within a substitution, precede it by a

backslash.) You can include a total of 256 substitutions in this way, spread in any way between the *varsub$_x$* parameters.

status will be set to 0 to indicate that a page was successfully generated (although it may still fail to meet the necessary syntactic requirements for a valid webpage). –1 indicates that *file* could not be opened.

A common mistake in creating a template web page is to forget to include the "content-type" header line, which it not required in static web pages but is required in pages generated through CGI. As a minimum, the template and/or resulting page should contain the following elements (note the mandatory blank line following the "content-type" header:

```
content-type: text/html

<HTML>
<HEAD>
<TITLE>This is the page title</TITLE>
</HEAD>
<BODY>
bla bla bla
bla bla bla
</BODY>
</HTML>
```

The *varsub#* parameters must be null terminated, and the total length of any line of the file, after substitutions, will be limited to 1024 bytes! (Generally speaking, line terminations mean nothing to web browsers, so it is unlikely that there is any need for your web page source file to have such long source lines.)

The variable substitution NAMEs are case sensitive, but do not have to be terminated or delimited in any special way. To avoid inadvertent matches, you might want to prefix and/or suffix your NAMEs with some special character, like $.

As of build 837, *file* may contain an AMOS or native file specification. (To force an otherwise ambiguous specification to be interpreted as native, precede it with the "./" (which refers to the current directory), e.g. `"./NativeFile.html"`. Prior to build 837, it was always interpreted as a native specification, which among other things meant that it was case sensitive.

This operation can be difficult to debug, since if the resulting page is not valid, the web server will not display any of it and it will not be available for inspection. A first debugging step would be to attempt to launch *file* in the browswer directly, which should work (displaying the variable names instead of the replacement values) provided that variables are not used for critical elements of the HTML page structure. A second debugging step would be to add TRACE=XCALL, or otherwise arrange for the command SET TRACE XCALL ON to be executed before the program runs. This will not only log the XCALL parameters (for all XCALLs) to the ashlog.log file, but will also create a file CGIUTL.LOG containing a copy of the merged file that was output to *stdout*.

Opcode 4

```
xcall CGIUTL, 4, string, status
```

opcode 4 writes *string* to *stdout*. Use this in place of PRINT statements to output individual characters or lines to the new web page. (In -cgi mode, the output of PRINT statements is sent to the *stderr* stream rather than *stdout*. See the *A-Shell Setup Guide* for further notes on the -cgi switch.)

Opcode 5

```
xcall CGIUTL, 5, envvar, envdef, status
```

opcode 5 retrieves the contents of the environment variable *envvar* into the string variable *envdef*. *status* will be set to 0 for OK, or 1 if the variable had to be truncated to fit in *envdef*. If there was no such variable defined, *envdef* will be null.

```
xcall CGIUTL, 6, file, status, outch {,varsub1 {,varsub2 &
     {,varsub27}}}
```

Opcode 6

opcode 6 is identical to *opcode* 3 except that instead of outputting to stdout, it outputs to file channel *outch* (which must have been opened for output).

This function is quite useful for expanding template files containing variables to be replaced with data, which applies to a variety of activities not necessarily related to CGI, but was grouped here because it shares the same logic with the existing CGIUTL opcode 3.

CHKONE

```
xcall CHKONE, c$
```

CHKONE.SBR is similar to TINKEY in that it allows you to check if a keyboard character is available, without getting stuck waiting if there is no character available. However, unlike **TINKEY**, which returns the actual character if available, **CHKONE** returns a "0" if no character is available, else "1". Any available characters remain in the input buffer, waiting for you to use another input routine to get them. c$ is a one-byte string.

CMDR

```
xcall CMDR
```

Calling **CMDR.SBR** has the same effect as if the line **:R** were executed in a command file. It can be particularly useful when trying to integrate command files and BASIC programs in order to implement more .LIT A-Shell commands.

CMDR.SBR has no effect if the command file is currently in either **:R** or **:T** modes.

COMIO

```
xcall COMIO, opcode, ch, buffer, status, count
```

COMIO.SBR is included with A-Shell/Windows (32-bit version only) to allow simple I/O on Windows serial ports (since you cannot open up a Windows serial port as a file, like you can on other platforms).

Parameters

opcode

is a numeric variable (any type) which specifies the operation:

Opcode	Meaning
1	Open port specified in the *buffer* parameter, return channel in *ch*
2	Close port specified by *ch*
4	Check if any characters available to read; return count in *count*
8	Input a maximum of *count* characters (if –1, input a line)
16	Write exactly *count* characters (or up to null if *count*=0)

ch

is a numeric variable (F or B) which specifies the channel number associated with the port. This value is returned from the open operation, and must be passed to the subroutine for all other operations.

buffer

is usually a string variable to pass the data to be sent or received (for read/write operations). For the open operation, it is used to pass control information about the port settings, and must be mapped as follows:

```
MAP1 BUFFER
   MAP2 BYTESIZE,B,1     ! 7=7 bits, 0 or 8 = 8 bits
   MAP2 PARITY,B,1       ! 0=none, 1=odd, 2=even, 3=mark, 4=space
   MAP2 STOPBITS,B,1     ! 0=1, 1=1.5, 2=2
   MAP2 FLOWCTL,B,1      ! Add: 1=DTR/DSR, 2=RTS/CTS, 4=XON/XOFF
   MAP2 MISC,B,2         ! Unused
   MAP2 PORT,S,20        ! e.g. "COM1" (no colon)
```

status

is a floating point variable which returns a status code indicating the level of success or failure:

Status	Meaning
0	Success
-1	Parameter error
-2	No more memory handles available
-3	Memory allocation failure
-4	Unable to create write event
-5	Unable to create read event

Status	Meaning
-6	Invalid channel passed in CH argument
-7	Unable to open connection to port
-8	Control-C received while waiting for input
>0	Misc system error (errno)

count
is a floating point variable whose usage depends on the operation, as given in the table below:

Operation	Usage of COUNT parameter
Open (1)	Must be set to the desired baud rate.
Close(2)	Ignored
Check(4)	Returns number of characters available to be read.
Read(8)	Must be set to the number of characters you wish to read. If fewer characters are available, the routine will wait. Two special flag values may be used for variable length line-oriented input. If set to 0, the routine will input up to a CRLF pair (waiting as necessary) and then discard the CRLF (similar to INPUT LINE). If set to –1, the routine will input up to a CR, and then discard the CR. In all cases, on return, it will contain the number of characters received (including any discarded line terminators).
Write(16)	Must be set to the number of characters to write (from the BUFFER parameter). You may set it to 0 to output up to the first null byte in the BUFFER string. On return, it will contain the number of characters actually output.

Comments

Refer the sample program, COMIO.BAS, which is included with the standard release, for more information.

For more sophisticated applications involving serial communications, Autolog from Soft Machines is recommended.

COMMON

```
xcall COMMON, send, msgnam, var

xcall COMMON, recv, msgnam, var
```

COMMON.SBR is a widely used (and widely modified) routine for sending a block of data from one program to another.

Parameters

```
MAP1 SEND,B,1,0
MAP1 RECV
   MAP2 F'RCV,B,1,1
   MAP2 RCVFLG,B,1,0
```

send, recv
must be mapped as shown above. As the names imply, *send* is used for sending a packet and *recv* for receiving one.

rcvflg
will be returned as 0 if the requested packet is not found, else it will be set to a non-zero value.

msgnam (string, up to 6 characters)
specifies the name of the packet. This is how packets are identified if more than one is being stored at the same time.

var (unformatted)
supplies the data to send or returns the received data. See below for size limitations.

Comments

Under AMOS, normal usage requires that you load **COMMON** into user memory, since it stores the data within itself. **COMMON** does not exist as a separate module under A-Shell; however, the command line `.LOAD BAS:COMMON` will have the same effect as under AMOS, in that all common packets will be cleared.

The default packet set-up is 6 packets of 150 bytes. This may be changed by adding, for example, the following line to the MIAME.INI configuration file:

```
SBR=MSGNUM:4,MSGSIZ:1024
```

This would decrease the number of packets to 4, with a size of 1024 bytes each. More information is given in the documentation for the MIAME.INI file.

As with the standard version of **COMMON** under AMOS, a read operation is destructive. That is, the packet is cleared upon reading it. Thus, if you want to read the packet and then pass it on to another program, you have to rewrite it after reading. Some people use a modified version of COMMON which has a non-destructive read. To support this variation, add SBR=COMMONNDR to your MIAME configuration file (the NDR stands for *Non Destructive Read*).

CONDEV

```
xcall CONDEV, dev$ {,dns'flag}
```

CONDEV.SBR returns an identifier for the client workstation or console/terminal device. It is useful in situations where you want to identify the physical workstation, rather than the user or session. The identifier can take one of several forms, depending on the operating system and other factors. Under Windows, it will return the NetBIOS machine name (e.g. "SALESPC"). Under UNIX, it will first check to see if the environment variable REMOTEHOST is defined, and if so, it returns its definition. (Many shells will define the environment variable REMOTEHOST to contain the IP address of the workstation). Otherwise, if it can determine the IP address of the client workstation, it will return that in ###.###.###.### format. Otherwise, it will return the terminal device (pseudo or real tty) name, e.g. "ttya02" or "pts/0". If the *dns'flag* parameter is specified and non-zero, an attempt will be made to convert the IP address to the equivalent host name. This can sometimes introduce a delay if there is no local DNS server.

See GETUSN, which is similar.

If you are using the ZTERM terminal emulator (and possibly others) you can retrieve the client's IP address using an ESC sequence; see the documentation on ZTERM Escape Sequences.

CRC16

```
xcall CRC16, block, crc, size
```

CRC16.SBR calculates a 16 bit CRC (aka CRC-16) for the specified block of data.

Parameters

block (numeric type)
is an unformatted variable of arbitrary size.

crc
will return the CRC-16 value.

size
specifies the number of bytes within *block* to consider.

DATES

DATES.SBR performs several different date utility functions, and has several calling formats:

Convert from one format (DATE1) to another (DATE2):

```
xcall DATES, 1, result, date1, date2
```

Compute date (DATE2) which is a specified number of days (DAYS) after DATE1:

```
xcall DATES, 2, result, date1, date2, days
```

Compute numbers of days DATE2 is after DATE1:

```
xcall DATES, 3, result, date1, date2, days
```

Compute DATE2 based on year, week, day of week in X,6 or X,5 format of DATE1:

```
xcall DATES, 4, result, date1, date2
```

Compute DATE2 based on year, month, week, week and day of week in X,6 format of DATE1:

```
xcall DATES, 5, result, date1, date2
```

Compute DATE2 as last day of month in DATE1:

```
xcall DATES, 6, result, date1, date2
```

Output formatted date (similar to ODTIM):

```
xcall DATES, 7, result, date, time, flags, outbuf'or'chan
```

Input formatted date (similar to IDTIM):

```
xcall DATES, 8, result, inputstr, flags, outdate, outtime
```

Return DATEOK (F) <>0 if DATE is between LODATE and HIDATE (inclusive):

```
xcall DATES, 9, result, date, lodate, hidate, dateok
```

The *result* variable (any B or F data type) will return a code indicating if the operation was successful or what kind of error occurred:

Value	Meaning
0	OK
1	Function number out of range (1-9)
2	Error in conversion of input date
3	Invalid format for date
4	Improper number/type of parameters
5	Invalid format for *days*
6	Error locating file channel (*op* 7)

The date parameters, which are listed above as *date, date1, date2, lodate* and *hidate* can be mapped in any of the following ways:

```
MAP1 ADATE,S,8                ! MM/DD/YY format
MAP1 XDATE                    ! Separated format (3 to 7 bytes)
    MAP2 MONTH,B,1
    MAP2 DAY,B,1
    MAP2 YEAR,B,1
    MAP2 DOW,B,1              ! 1=Monday...7=Sunday (optional)
    MAP2 YWEEK,B,1            ! Week of year (1-52)     (optional)
    MAP2 MWEEK,B,1            ! Week of month (1-5)     (optional)
    MAP2 MDAYS,B,1            ! Days in month (1-31)    (optional)
MAP1 IDATE,B,4                ! Internal true Julian
MAP1 BDATE,B,3                ! AlphaBASE (same as 3 byte XDATE
MAP1 CDATE,B,2                ! Century Julian (days since 1/1/1900)
MAP1 JDATE,B,2                ! Yearly Julian (days since start of yr)
MAP1 FDATE,F,6                ! Special case (0=today on input)
```

The MIAME.INI parameter SBR=CCYY:## is used to determine the century cutoff on input dates in MM/DD/YY format. See the discussion of the SBR parameter in the *A-Shell Setup Guide* for more details.

DELCHR

```
xcall DELCHR, chrlst, string
```

DELCHR.SBR removes all the characters that are specified in the string variable *chrlst* from the string variable *string*. See the BASIC Plus function EDIT$ for related functionality.

DEVCHK

```
xcall DEVCHK, dev, status
```

DEVCHK.SBR can be used to check whether an AMOS logical device is defined to A-Shell. The *dev* parameter should contain the name of the device (e.g. DSK0); on return the *status* parameter (F,6) will be set to 1 if the device exists or 0 if it does not.

DSKPPN

```
xcall DSKPPN, disk, ppn

xcall DSKPPN, a$, ppn2
```

DSKPPN.SBR retrieves either the disk and PPN, or just the PPN. In the second format, it mimics a custom subroutine used by some people under AMOS, called **GETLOG.SBR**. In that case, use ALIAS=GETLOG:DSKPPN.

Parameters

```
MAP1 Disk,S,6     ! (e.g. "DSK01")
MAP1 PPN,S,7      ! (e.g. "7,13")
MAP1 PPN2,S,10    ! (e.g. "[7,13]")
MAP1 A$,S,1       ! (dummy parameter)
```

DSTOI

```
xcall DSTOI, sepdate, jdate
```

DSTOI.SBR converts a separated date (which is the format returned by the DATE system function in BASIC) into a Julian date which starts with 1/1/1900.

Parameters

```
MAP1 Sepdate
   MAP2 Mon,B,1     ! Month 1-12
   MAP2 Day,B,1     ! Day 1-31
   MAP2 Yr,B,1      ! Year-1900
   MAP2 Dow,B,1     ! Day of week (0=Mon, 6=Sun)
MAP1 Jdate,F         ! # days since 1/1/1900
```

You can use ODTIM or DATES to convert the Julian date back to various separated and printable formats.

ECHO

```
xcall ECHO {,channel}
```

ECHO.SBR is used to turn terminal echo back on after disabling it with the NOECHO subroutine. It may also be used to return a serial port under UNIX to its normal settings. See the descriptions of GET and NOECHO.

EMAILP

```
command = sbx:EMAILP {,flags {,subject {,intro}}}
```

(Windows only) **EMAILP.**Error! Reference source not found. is a print filter subroutine intended only for use in a COMMAND statement within a printer init file. It launches your local email client and embeds the specified printfile into the body of the message, allowing the user to then address and complete the sending process.

Note that since the way this subroutine can be called is from within a printer init file, the parameters must be specified as literals. (There are no "variables" within printer init files.) Remember to enclose strings in quotes if they contain embedded commas.

Parameters

flags

may optionally be specified as the sum of one or more of the following values:

Value	Meaning
1	Receipt requested
2	Send file as attachment instead of embedded within message. (Not yet implemented as of March 2003; consult source code EMAILP.BAS for update notes.)
4	Use HTML coding to set fixed pitch (when embedding)

subject

may optionally be specified as a literal string containing the desired subject of the email message.

intro

may optionally be specified as a literal string containing a string of introductory text that will be inserted at the top of the message.

ERRMSG

Documentation update 16 Nov 04: Beginning in Build 892 of 10 June 04, **ERRMSG** (i.e., DERR) now uses a Windows-style message box if GUI support is available.

```
xcall ERRMSG {,errarg}

xcall ERRMSG, msg$, errnum

MAP1 ERRARG
   MAP2 ERR'ID,S,6      ! (this field is ignored)
   MAP2 ERR'CDE,B,2     ! app-defined error #
   MAP2 ERR'MSG,S,50    ! app-defined error message
```

ERRMSG is a handy way to handle BASIC error trap reporting. Add the subroutine to your BASIC error trap, and it can display a standardized message at the bottom of the screen (listing information about the error) and log the error to file, and/or retrieve the text associated with the current error. The routine must be set up as an alias to DERR in the MIAME.INI file as follows:

```
ALIAS=ERRMSG:DERR
```

> DERR is a Debug plc routine which is similar to ERRMSG. It is not documented here since we prefer that you use ERRMSG, but if you are already using DERR, feel free to continue, but note that it expects four arguments, rather than the 0-2 that ERRMSG expects.)

If 0 or 1 argument is passed and a BASIC error has occurred, then ERRMSG ignores the *errarg* argument (if passed) and displays a three line message at the bottom of the screen relating to the BASIC error, which looks something like this:

```
* ERROR MESSAGE * TSKAAA    Illegal record number
Error 31 in Line 360 of MYPROG Last File: 5 (ERRMSG.TMP)
Press ESC to Abort
```

You may also call the **ERRMSG** directly (i.e. absent any BASIC error) to report application-defined errors. In that case, you must include the *errarg* parameter, assigning your own values to the fields. For example, consider the following code:

```
125   ERR'CDE = 142
130   ERR'MSG = "Something bad has happened!"
135   xcall ERRMSG,ERRARG
```

This would display the following message (where "MYPROG" is the current program name):

```
* ERROR MESSAGE * TSKAAA    Something bad has happened!
Error 142 in Line 135 of MYPROG
Press RETURN to Continue or Retry
```

In either of the above cases, the subroutine will also log a similar message to the text file BAS:BASERR.LOG.

If two arguments are passed (as in the second sample format above) then it simply retrieves the BASIC error message associated with the specified error #. This can be useful for creating your own error message display

within an error trap, since when error trapping is enabled, BASIC no longer displays the error message. (It only supplies you with the error number, last line number, and last channel number, via the ERR(0), ERR(1), and ERR(2) functions.)

EFS

This subroutine was added to A-Shell in build 889 of 25 May 2004.

(Linux) This feature has been added to A-Shell for a single developer but which will likely be needed by others in the future. Basically it allows files to be encrypted and decrypted on command (via an XCALL) using AES encryption. Perhaps more importantly, you can read and write directly to encrypted files just as if they were normal files. The objective is to provide a higher degree of privacy protection for sensitive data by closing three of the main security gaps in most organizations: 1) hackers breaking into the network and stealing data files directly; 2) employees or other insiders copying data from the server to the their PC and then to email or removable media, and 3) stealing backups.

The new **EFS.SBR** handles most of the operations related to encryption:

```
xcall EFS,0,STS                ! check if EFS is available
xcall EFS,1,STS,KEY            ! specify new encryption key
xcall EFS,2,STS,FSPEC          ! check if FSPEC file encrypted
xcall EFS,3,STS,ISPEC{,OSPEC}  ! encrypt ISPEC {into OSPEC}
xcall EFS,4,STS,ISPEC{,OSPEC}  ! decrypt ISPEC (into OSPEC}
```

STS (F,6) is returned as follows:

```
 0 = success
-1 = EFS not available
-2 = EFS not licensed
-3 = param error
>0 = errno
```

KEY (X,32) should be mapped as follows

```
MAP1 KEY,X,32
MAP2 KEY$,S,32,@KEY
```

By specifying the unformatted KEY parameter, it will not show up in the trace log even if TRACE=XCALL is set. Furthermore, if your key is less than 32 bytes long, the remaining key bytes will be supplied from a default internal ashell key. By taking advantage of this feature, even if your part of the key was exposed and a copy of the file stolen, the culprit would still need a licensed copy of A-Shell to decrypt it.

```
ISPEC, OSPEC (string) are AMOS or native filespecs.
```

When the EFS license option in enabled, A-Shell will automatically detect when a random, ISAM, or ISAMPLUS file has been encrypted and thus there is no need to specifically identify to A-Shell which files are encrypted, except with you create a new file. In that case, for RANDOM files created with ALLOCATE, use XCALL ASFLAG,512 prior to the allocate to set the encryption flag. (As with other ASFLAG values, the setting only lasts until the end of the current program.) For ISAMPLUS, you can use the new ISMUTL /E switch (requires ISMUTL.LIT 1.3(128) or higher). For old ISAM, you can first use ISMBLD, then use XCALL EFS,3,STS,ISPEC to encrypt the DAT and/or IDX file(s). (An ISMBLD switch may be added if there is a demand.)

Note that direct reading and writing of encrypted files only works for “contiguous” files (random, ISAM, ISAMPLUS). For printfiles, if you want to encrypt them, you will probably need to use XCALL EFS after

closing (and after spooling, if applicable) the file. Similarly, you would use XCALL EFS again to decrypt it before accessing (viewing, reprinting) the file later.

EZSPL

EZSPL.SBR is a very useful utility, which may be used in place of SPOOL, being fully upward compatible. (Indeed, under A-Shell, SPOOL is implemented as an alias to EZSPL.) It provides three main functions:

- EZSPL incorporates a **screen preview facility**, enabling the user to browse through the document either before or instead of sending it to a printer. This feature has been fully implemented under A-Shell. In fact, the A-Shell version of the viewer (known as EZ-TYPE or, under it's latest incarnation, EZ-VUE) supports many enhancements over the AMOS version.
- EZSPL can provide a **front-end printer selection**, enabling the user to select the printer and various print options (such as banner, header, etc.). Many of the switches are specific to the AMOS environment, and so there is a fair amount of divergence here between the AMOS and A-Shell versions, but the concepts are similar and the differences can be handled in the configuration files, external to the application source code.
- EZSPL can aslo provide **transparent character translation** through what it terms *port filters*. These enable programs written to generate escape sequences for specific printers to have their codes transparently modified to the sequences required for other printers as the documents are output.

Port filters are not implemented under A-Shell, but the equivalent functionality is available using either the COMMAND=SBX:<routine name> option in the printer initialization file, or traditional UNIX filters in the UNIX environment.

In addition to the above features, EZSPL, like SPOOL, performs the mundane task of sending the specified file to the specified print queue. Two calling formats are available, each described below, followed by the a brief discussion of the EZ-SPOOL configuration file processing.

Old format

```
xcall EZSPL, file {,printer {,switches {,copies {,form {,lpp {,width
      {,prefix {, suffix}}}}}}}}
```

Parameters

file (string)
is the specification of the file to print, and is the only required parameter.

printer (string)
is the name of the printer to send the file to. If not specified, the default defined by the PRINTER statement of the MIAME.INI file will be used. In most cases, the *printer* name should correspond to a printer initialization file with a matching name, either SYS:<printer>.INI or ASHCFG:<printer>.PQI. The exceptions to this rule are as follows.

Under Windows, you may specify the pseudo printer name "PROMPT", which is interpreted as a printer whose initialization file contains the following:

```
DEVICE = PROMPT:
PASSTHROUGH = OFF
```

```
PITCH = AUTO
```

If the name you specify is longer than 6 characters (which was the maximum for AMOS printer names and remains the maximum for A-Shell printers which have corresponding initialization files), then it is interpreted like the PROMPT example above, except with the DEVICE set to the name you specify. (This "implied printer" technique can be taken to its logical extreme by specifying a \\machine\sharename name for the printer, in which case there doesn't even need to be a local printer driver definition.)

Under UNIX, if you specify a printer name which has no corresponding initialization file, then it is treated as if the initialization file were simply:

```
DEVICE = <printer>
```

Refer to the printer configuration documentation for more information about printer initialization files.

switches (numeric)
may be any combination of the following:

Value	Equivalent PRINT.LIT switch
1	BANNER (applies only to UNIX, if at all)
2	NOBANNER
4	DELETE
8	NODELETE
16	HEADER (ignored by A-Shell)
32	NOHEADER (ignored by A-Shell)
64	FF (formfeed after printing)
128	NOFF
256	WAIT (ignored by A-Shell)
1024	INFORM (ignored by A-Shell)
2048	KILL (ignored by A-Shell)
8192	PASSTHROUGH (Windows only)
16384	NOPASSTHROUGH (Windows only
32768	LANDSCAPE (Windows only)
65536	NOLANDSCAPE (Windows only)

Any switches specified this way will override the corresponding switches in the printer's initialization file.

Note that the while the LANDSCAPE switch will force landscape mode regardless of the orientation currently set in the printer driver's properties, this is not currently true of the NOLANDSCAPE switch. In other words, if the user changes the printer's orientation to landscape in the printer properties then A-Shell will not be able to change that. But as long as the printer driver's orientation is left at portrait, A-Shell will be able to print in either portrait or landscape mode. The only point of the NOLANDSCAPE switch then is to override the ORIENTATION=LANDSCAPE statement in the printer init file.

copies (numeric)
is the number of copies to print (default 1).

form (string)
is the form name. Note that names are not generally used in the Windows or UNIX world and are thus ignored by A-Shell. To get the equivalent effect, you would probably define multiple logical printers associated with a single physical printer.

lpp (numeric)
is the number of lines per page (ignored).

width (numeric)
is the number of columns per page (ignored).

prefix (string)
optionally specifies the name of a file to be prepended to the front of the printfile before sending it to the printer. This is useful for sending printer initialization codes or routing instructions.

suffix (string)
optionally specified the name of a file to be appended to the end of the printfile.

New format

xcall EZSPL (or spool), file, table

The *table* parameter layout is as follows. Fields marked with an asterisk are ignored by all A-Shell platforms.

```
MAP1 TABLE
    MAP2 PRINTER,S,64         ! spooler name (was 6 under AMOS)
    MAP2 CPU,F,6              !*CPU id
    MAP2 SWITCHES,F,6         ! option switches
    MAP2 COPIES,F,6           ! copies to spool
    MAP2 BANNER,S,50          !*banner
    MAP2 LPP,F,6              !*lines per page
    MAP2 WIDTH,F,6            !*columns per page
    MAP2 FORMS,S,6            ! printer form
    MAP2 PRI,F,6              !*priority
    MAP2 ADATE,B,4            !*spool after date
    MAP2 ATIME,B,4            !*spool after time
    MAP2 RESTART,F,6          !*restart
    MAP2 START,F,6            !*start page
    MAP2 FINISH,F,6           !*finish page
    MAP2 LIMIT,F,6            !*limit form feeds
    MAP2 OPTIONS,F,6          ! (EZ-SPOOL) options
    MAP2 SEQNO,F,6            !*(EZ-SPOOL) sequence #
    MAP2 COUNT,F,6            !*(EZ-SPOOL) # blocks queued
    MAP2 ITCERR,F,6           !*(EZ-SPOOL) ITC error
    MAP2 NONITC,F,6           !*(EZ-SPOOL) spooler
    MAP2 FILERR,F,6           !*(EZ-SPOOL) file error code
    MAP2 EXTOPT,F,6           ! (EZ-SPOOL) extended options
    MAP2 PREFIX,S,28          ! (A-Shell) prefix file
    MAP2 SUFFIX,S,28          ! (A-Shell) suffix file
    MAP2 HOSTROWS,F,6         ! (A-Shell) # of rows reserved for host in EZTYP
    MAP2 EZCOLORS             ! User-defined color scheme
        MAP3 EZFG'TXT,B,1     ! text colors
```

```
MAP3 EZBG'TXT,B,1     !
MAP3 EZFG'BDR,B,1     ! border colors
MAP3 EZBG'BDR,B,1     !
MAP3 EZFG'CMD,B,1     !
MAP3 EZBG'CMD,B,1     ! command colors
MAP3 EZFG'STS,B,1     !
MAP3 EZBG'STS,B,1     ! status colors
MAP3 EZFG'HLP,B,1     !
MAP3 EZBG'HLP,B,1     ! help colors
MAP3 EZFG'HLT,B,1     !
MAP3 EZBG'HLT,B,1     ! highlight colors
MAP3 EZFG'MNU,B,1     !
MAP3 EZBG'MNU,B,1     ! menu colors
MAP3 EZFG'BRF,B,1     !
MAP3 EZBG'BRF,B,1     ! brief colors
```

The fields marked (EZ-SPOOL) were not part of the original AMOS specification and were added by EZ-SPOOL (as developed for AMOS). The fields marked (A-Shell) are A-Shell extensions that were not part of either the original AMOS or EZ-SPOOL specifications. All of the fields for both types of extensions are optional under A-Shell.

The PRINTER field was 6 bytes under AMOS and up until 4.7(827) of A-Shell. We expanded at that point to allow for the possibility of using UNIX and especially Windows printer names (which can be quite long.) If you use the 64 byte version of the PRINTER field, then you MUST specify all of the other fields (even the ones which we just got through saying were optional.) A-Shell can only determine which version of PRINTER you used by considering the total size of the TABLE parameter.

The EZCOLORS fields, if specified, will override the normal default color scheme used by EZ-SPOOL. The only exceptions to this are that they will be ignored if they are ALL ZERO, and that that can be overridden themselves by the EZCLR section of the INI.CLR file. See the *A-Shell Setup Guide* for details on the color configuration file format, and also see MIAMEX 116: Process color ini file.

The *options* parameter above may be set to any combination of the values in the following table. Values that are ignored by A-Shell have been omitted (which is why there are gaps).

Option	Meaning
1	PRINTER selection menu on
2	PRINTER selection menu off
4	OPTION menu on (questions at bottom of PRINTER menu)
8	OPTION menu off
16	TYPE (preview) allowed (by selecting printer X)
32	TYPE (preview) not allowed
131072	ASKPRT on (Prompt to confirm user wants to print the file)
262144	ASKPRT off
33554432	BRIEF on (single line printer selection prompt)
1747483648	Preview file only (same as XCALL EZTYP)

The *extopt* parameter above may be set to any combination of the values in the following table. (These may also be set via the EXTOPT=### parameter in the EZSPL Configuration Files (which see in Setup Guide)Values that are ignored by A-Shell have been omitted (which is why there are gaps).

Option	Meaning
4	Disable screen save/restore
32	Allow exit from EZVUE with left arrow
64	Allow exit from EZVUE with TAB
128	Allow exit from EZVUE with F1-F16
256	Don't ask questions on PRINT from within EZVUE
512	Hitting HOME when already at the top of the file in EZVUE acts like the PRINT command.

EZ-SPOOL Processing

The A-Shell version of EZ-SPOOL is quite similar to the AMOS version, but with some limitations and some extensions. Perhaps most importantly for those not interested in these features, they remain silent and hidden unless you go out of your way to activate them. Such people may safely skip this section. For the rest of you, we will give a very brief overview of EZ-SPOOL configuration, concentrating mainly on the areas of difference between the AMOS and A-Shell versions. Those wanting to know more about EZ-SPOOL should contact MicroSabio for a copy of the *EZ-SPOOL User's Guide*.

The first thing **EZSPL** (SPOOL.SBR) does upon being called is look for configuration files. The A-Shell version of the configuration file search path is much more abbreviated than the AMOS version, consisting only of the following:

```
EZ:<printfilename>.SFL
EZ:<programname>.SPG
EZ:<printername>.SPR
EZ:<termname>.SPL
MEM:SYSTEM.SPL
SYSTEM.SPL[p,pn]
SYSTEM.SPL[p,0]
EZ:SYSTEM.SPL
```

If none of these are found, then printing proceeds as if EZ-SPOOL did not exist. Otherwise, it opens the first file it finds and process the parameters contained within. The only parameters supported under A-Shell are:

```
MENU=<boolean>
TYPE=<boolean>
BRIEF=<boolean>
ASKPRT=<boolean>
DEFAULT=<boolean>
OPTIONS=<boolean>
AUTOWIDTH=<boolean>
WAIT=<# seconds>
PRTSPL=spooler1,description
PRTSPL=spooler2,description            (up to 64 PRTSPL statements)
```

Each of these statements must be in all upper case, with no spaces, except within a description field. (e.g. "`MENU = ON`" and "`menu = On`" are invalid). Boolean options are "`ON`", "`OFF`", "`YES`", "`NO`", "`TRUE`" and "`FALSE`".

Individual sections may also be added to the configuration file according to user name.

> User name is determined by the native operating system logon procedure, and can be seen in the SYSTAT display or retrieve with GETUSN. It is not case sensitive in this context.

For example, a configuration file may look like this:

```
MENU=ON
TYPE=ON
ASKPRT=ON
DEFAULT=JET
PRTSPL=JET,System LaserJet
PRTSPL=DRAFT,Oki draft

[jack]
MENU=OFF
TYPE=OFF
ASKPRT=OFF
DEFAULT=JET2
```

In this case, for all users but "jack", the MENU, TYPE, and ASKPRT options will be ON, the default printer will be JET, and a menu will be displayed showing the JET and DRAFT printer choices. However, if the user is "jack", there will be no menu, and the only effect of the configuration file processing will be to set the spooler name to JET2 if no spooler was specified in the XCALL SPOOL.

The configuration file is processed from the top down, and any statements which are not part of a [<username>] section are processed for all users. (The top section may also be given the heading [always].) Statements within a specific [<username>] section may then override prior statements in the top or [always] section.

This capability of adding sections for individual users allows a great deal of flexibility for large systems, but is not supported under the AMOS version of EZ-SPOOL. For those not familiar with EZ-SPOOL, here is a brief rundown on the configuration file commands. (All of these are also supported under EZ-SPOOL/AMOS):

MENU=ON causes a printer selection menu to appear. The choices on the menu consist of the printers defined in the PRTSPL statements.

ASKPRT=ON provides an option to not go through with the spooling operation. If MENU=ON, the operation is aborted by entering choice Z. If MENU=OFF, the user is prompted on line 24 of the terminal whether to print.

AUTOWIDTH=ON (default) causes EZTYP/EZVUE to automatically switch the display into 132 columns when a line longer than 80 columns is found in the first several lines of the file. If you find that annoying, you can set AUTOWIDTH=OFF to prevent it.

TYPE=ON provides an option to preview the file on the screen. If MENU=ON, this becomes choice X. Otherwise, the user is prompted on line 24 whether to preview the file. The preview utility allows you to page

forwards and backwards through the file. With the right video set-up, it will also allow you to switch between 80 and 132 columns using the left and right arrow keys.

BRIEF=ON activates a single-line menu (on line 24) asking for the name of the printer only. The user can TAB through the available choices. If OPTIONS=ON (or omitted) it will also prompt for the number of copies.

WAIT specifies the number of seconds that various EZ-SPOOL screen prompts will wait for an answer before using the default value. (0 for unlimited wait.)

DEFAULT defines the default spooler name to be used if no spooler name is passed in the XCALL SPOOL. This overrides the default established by the PRINTER= statement in MIAME.INI.

PRINTER (in the EZ-SPOOL configuration file, as opposed to MIAME.INI) is like DEFAULT but overrides the spooler choice specified in the XCALL. (The confusion with MIAME.INI PRINTER parameter is regrettable, but this terminology matches EZ-SPOOL under AMOS.)

PRTSPL=spooler, description (e.g. PRTSPL=JET, LaserJet) statements establish choices that appear on the printer selection menu. If MENU=OFF, these statements have no effect.

> Also see Printer Configuration in the A-Shell Setup Guide for more details on the EZSPL configuration files.
>
> The ability to send a file to the printer can be disabled via the GOP2_NOPSDLG options flag (see MIAMEX 60: Set OPTIONS flags); if set, an attempt to print will display the error "Insufficient privileges to print" (from ERRMSG.xxx 003,045).

EZTYP

```
xcall EZTYP, fspec {,topmgn {,leftmgn}}
```

EZTYP.SBR provides a simple way to call the text/print file viewer within the EZ-SPOOL subsystem. Note that it is possible to print individual pages or page ranges (including the entire file) within EZTYP.SBR, so it could in some circumstances be used in place of EZSPL.SBR. See EZTYP.LIT in the *A-Shell Command Reference* for more information on the capabilities of the print file viewer.

Note that EZTYP.SBR behavior may be affected by the WAIT (timeout) and AUTOWIDTH parameters in the EZSPL configuration files, as well as the EZCLR entry in the INI.CLR file. Refer to *Printer Configuration / EZSPL Configuration* and *Using A-Shell / General Topics / Color Customization*, both in the in the *A-Shell Setup Guide*, for more information.

The *topmgn* and *leftmgn* commands allow you to preserve the specified number of rows and columns at the left and top of the screen to be untouched by the EZTYP display.

F2HOST

```
xcall HOST2F, alphaflt, hostflt
```

F2HOST.SBR converts a 6 byte AlphaBASIC-format floating point value to the equivalent IEEE 4 byte (single precision) or 8 byte (double precision) floating point format. Although this is done internally by A-Shell all the time, since it uses the IEEE format for internal calculations, the subroutine may be handy when exporting data to outside sources.

Parameters

alphaflt
should be mapped as F,6 and will return the converted value.

hostflt
should be mapped as X,4 or X,8 depending on the size of the IEEE floating point value to be converted.

See HOST2F for the reverse conversion. Note that when exporting data, it is usually more practical to output in string CSV format. See the **WRITECD** and WRITETD statements in the *A-Shell Command Reference* for an easy way to do this.

FIFO

```
xcall FIFO, op, ch, buffer, flags, status
```

FIFO.SBR (UNIX only) provides a general-purpose technique for communicating with other processes, including those outside of A-Shell and possibly even on other machines (connected via a network) using named pipes. Named pipes (sometimes called FIFOs because that is the way they act) are supported under most implementations of UNIX and are similar to sockets, except that they are disk based and therefore somewhat removed from the details of networking.

Parameters

```
MAP1 OP, F
     ! opcode (any numeric type)
     ! 0 = create FIFO
     ! 1 = open FIFO
     ! 2 = close FIFO
     ! 3 = read FIFO
     ! 4 = write FIFO

MAP1 CH ,F
     ! channel returned from open; passed to other
```

```
     ! opcodes (any numeric type)

MAP1 BUFFER, S
     ! data used in various ways depending on
     ! opcode. May be type X or S as appropriate

MAP1 FLAGS, F
     ! misc. flags, based on OP (any numeric type)
     ! OP 0:  permission bits (modified by umask)
     ! OP 1:  open flags (0=read, 1=write, +4 for 'NDELAY'
     !       +100 for 'NONBLOCK' (see explanation below)
     ! OP 2:  ignored
     ! OP 3:  max # bytes to read (if 0, use map size of BUFFER)
     ! OP 4:  max # bytes to write (if 0, use map size of BUFFER;
     !if -1, use length of BUFFER up to first null)

MAP1 STATUS, F
     ! result code (any numeric type)
     ! OP 0:  0=OK, else -errno (see notes below on errno)
     ! OP 1:  >0=file number, else -errno
     ! OP 2:  none
     ! OP 3:  # of bytes read; else -errno
     ! OP 4:  # of bytes written; else -errno
```

Comments

NDELAY and **NONBLOCK** are mode options controlling how the routine attempts to read or write to a FIFO when the other end is not ready. The exact behavior may differ from one UNIX to another, so some experimentation may be required. In general, without either flag, read and write operations will suspend the caller until the operation can be completed. With **NDELAY** set, the operation will generally succeed and return immediately. That is, data written is buffered if the other end is not ready to read it, and read operations will just return NULL if no data is available. With **NONBLOCK** set, the routine will return immediately but will fail with an error code if the operation cannot be complete immediately.

> Under SCO OpenServer (and possibly others) **NONBLOCK** is probably not practical for the writing end of a pipe, and in all cases it is not practical for both ends to use **NONBLOCK.**

Error Codes: Aside from the standard UNIX **errno** values, two special STATUS values are possible:-999 indicates that you passed a bad OPCODE, and –998 is returned when in **NONBLOCK** mode and the operation could not be carried out immediately.

A sample program, FIFO.BAS, which demonstrates the use of this subroutine, is available from the Other Downloads page at the Microsabio website..

FLOCK

```
xcall FLOCK, action, mode, status, file, rec
```

FLOCK.SBR provides a scheme for implementing shared and exclusive file and record locks, with an option to wait or return an in-use code. The A-Shell version is fully upward compatible with the AMOS version, and in addition goes beyond the various 64K limits in the AMOS version to support file channels and record number values up to 2 billion.

Parameters

action

Numeric code indicating the action to perform:

Value	Description
0	Request permission to open file (see File)
1	Inform FLOCK that file has been closed
2	Program terminating – requests all locks on all files for this user to be released
3	Request permission to read record (see File, Rec)
4	Request permission to read/write all records of File
5	Release Rec (previously locked with Action 3)
6	Release File (previously locked with Action 4)

mode

Flags for exclusive and wait options:

Value	Description
0	Non exclusive lock, wait until request can be granted
2	Exclusive lock, wait until request can be granted
4	Non exclusive lock, no wait (return immediately with Status = 1 if request cannot be granted)
6	Exclusive lock, no wait (return immediately with Status = 1 if request cannot be granted)

status (F,6)

Return code (for actions indicated in parentheses):

Value	Description
0	Success
1	Resource unavailable (0,3,4)
2	Open request already granted (0)
3	Permission to open must first be granted (1,3-6)
4	Duplicate request for some record in file (3,4)

Value	Description
5	Permission to use record must first be granted (5,6)
100	Unimplemented action
101	File channel not open for random processing (3-6)
102	File channel already open for ISAM (0)
103	Less than 15 queue blocks available (0,3,4)
104	Illegal record (3-6)

file
Numeric value indicating the file channel

rec
Numeric value indicating the record number

Comments

FLOCK.SBR uses the queue block system to store the locks. You can specify the number of available queue blocks, and the location of the queue file (on disk or in memory) using the QUEUE= statement of the MIAME.INI. Note that since the memory option (available under UNIX/Linux only) is much faster, it is highly recommended if you are using FLOCK or XLOCK.

The **QUTL.LIT** utility can be used to display the locks in use and to manually clear locks left by an aborted job.

A-Shell's implementation of **FLOCK.SBR** has an additional feature in that after waiting for more than 10 seconds for a lock, it will write out a "pending" lock record to the queue. This has no effect except that it can be displayed in the QUTL utility (which can also be executed under program control to create a list file) and thus allows for possibility of reporting to the user who has the lock that he or she is waiting for.

The MIAME.INI option TRACE=LOCKS causes information about the conflicting lock that you are waiting for to be displayed on the bottom status line of the terminal.

FORCE

```
xcall FORCE, job, string {,status}
```

FORCE.SBR is a programmatic equivalent to the FORCE.LIT user command, both of which provide a scheme for forcing characters into another job's input stream.

FORCE relies on the same mechanism as SEND, which see for notes on limitations specific to particular operating system environments.

GDIPRT

```
command = sbx:GDIPRT {,<option1> {,<option2> {,<option3>...}}}
```

(UNIX/Linux only.) **GDIPRT.**Error! Reference source not found. is a print filter subroutine intended only for use in a COMMAND statement within a printer init file, which performs the function of transferring the file from the UNIX/Linux server to the workstation, where it be will be resubmitted to an AshLite printer of the same name (as the printer originally requested on the server). The usual motivation would be to take advantage of Windows printing features that would otherwise be unavailable on a UNIX or Linux server.

The standard version of **GDIPRT** requires that the workstation have a network connection to the server, using the ZTERM 2000 terminal emulator, as it uses ZTERM ESC sequences to perform an FTP file transfer. (You could modify the source code to work for other terminal emulators or other file transfer methods.)

The procedure works as follows:

- The file is submitted to the print on the server (say, to printer "prt1").
- The printer init file (e.g. ASHCFG:PRT1.PQI or SYS:PRT1.INI) invokes the **GDIPRT** subroutine via the COMMAND statement as shown above.
- The **GDIPRT** subroutine first attempts to detect if the terminal emulator is ZTERM, it aborts with STATUS=-2.
- It then opens the print file and scans it, looking for //IMAGE and //METAFILE statements. For any that are found, the corresponding files are transferred to the PC's temp directory (see <dir> option in the table below.)
- The print file itself is then transferred to the PC.
- AshLite is launched on the PC, passing it the command PRINT PRT1=<print file>. (Note that the printer name, in this case PRT1, will be the same as for the original print request.) If the AshLite launch fails, it will return STATUS=-3.
- Otherwise, we assume that the operation succeeded and return STATUS=0 (so that we don't try to print it on the server.)

The *option* parameters may be used to specify any of the following options (expressed as quoted, literal strings):

Value	Meaning
"VERBOSE"	Causes a dialog box to appear during the transfer operation showing details of the various steps. Useful for debugging, or just to have something to look at.
"DEBUG"	Causes AshLite to be launched with a visible window, and to not automatically exit on completion of the print command (thus making it easier to see what may have gone wrong). If VERBOSE was also specified, then the subroutine (on the server) will wait for an acknowledgment from the user before removing the display box.
"LOG"	Causes the subroutine to log details about the operation to the log file OPR:GDIPRT.LOG

Value	Meaning
"<dir>"	(Any string containing a backslash or colon, e.g. “C:\TEMP”) will cause the routine to use the specified directory as the temporary working directory on the PC. Otherwise, if ZTERM build 143+ is detected, it will use the %TEMP% environment variable, else it will use “C:\TEMP”

GET

```
xcall GET, buffer {,chan {,bytes'requested, bytes'received
      {,timeout}}}
```

GET.SBR is useful for simple or sophisticated input of raw characters from the keyboard, a file, or a UNIX device.

Parameters

buffer (string)
receives the byte(s) input. Note that no trailing null byte is added by the subroutine, so you may want to pre-clear this variable before calling **GET**. In the simplest case of **GET** where no parameters beyond *buffer* are specified, it will input a single character from the keyboard, waiting as long as necessary.

chan (optional, numeric)
File channel number of a file that has been opened for input, and from which the character(s) are to be read. If not specified, or zero, input is from the keyboard.

bytes'requested (optional, numeric)
Number of bytes to input. If not specified, default is 1. You can specify 0 in conjunction with the timeout feature if you just want to know if a character is available without actually inputting it.

bytes' received (Optional, numeric)
Returns the number of bytes that were actually input (and returned into the Buffer parameter). Normally this will be the same as *bytes'requested*, except in the following circumstances:

If *bytes'requested* was 0, then *bytes'received* will be 1 if there was one or more characters available to be input, or 0 to indicate no characters available.

If we hit the end of the input file, or the *timeout* period expires before reading the number of requested bytes, then *bytes'received* will be less than *bytes'requested*.

If an I/O error occurs, or if a *timeout* period is specified and no characters are input because the file was already at the end-of-file mark, *bytes'received* will be set to -1. (This distinguishes the case of end-of-file from timeout.)

timeout (optional, numeric)
Number of milliseconds to wait for input before aborting. If set to 0 or not specified, there is no timeout. Note that the actual elapsed time may be substantially longer that the specified period. This is both because of CPU time dedicated to other processes and because the total timeout period is actually divided into slices equal to the number of characters to be input, with each slice being used up only if no characters are available.

For example, if you have a 5000 millisecond (5 second) *timeout* and are requesting 5 characters, then you will actually be allotted 5 one-second timeout periods, and each period is only spent if no characters are input during that period. If exactly 1 character arrived every 0.75 seconds, then you would still have 5 unused seconds after 4 characters had be input, even though 3 seconds had actually

elapsed. This scheme may seem a bit strange, but is actually a reasonable compromise between the two extreme approaches of treating the timeout period as an absolute maximum or of resetting it entirely after each character.

When inputting from the keyboard, NOECHO will be set automatically (and left that way on exit).

Function key translations are not processed by GET.SBR. Use GETX or ACCEPT if you need that feature.

Inputting from a serial device (UNIX/Linux only)

To input from a serial device under UNIX or Linux, open it as an input file, e.g.:

```
OPEN #9, "/dev/cua0", INPUT
```

Then pass the file channel to XCALL GET just like any other input file channel. Beware, however, that under UNIX, such serial ports will be by default set to "line input" mode. Thus, while you may be asking for only 2 characters, you would not actually "see" them until a line terminator (usually LF) had been received. To avoid this annoyance and get the characters immediately as they are transmitted from the other end, you need to take the port out of "line input" mode. One way to accomplish this is to use XCALL HOSTEX with the UNIX *stty* command to turn off "icanon" mode, for example:

```
xcall HOSTEX, "stty -icanon < /dev/cua0"
```

Comments

Use the *stty* command only after opening the file channel. Otherwise, the settings may be wiped out by the open operation. Also, note that *stty* and its arguments are case sensitive, usually all in lower case.

You may also want to adjust other port settings, such as flow control, baud rate, etc. Consult the *stty* documentation for details on the available settings.

You may need to use *chmod 666* on the port device file to allow it to be read and its settings modified by a non-root user.

GETADR

```
xcall GETADR, var, varaddr {, varsize}
```

GETADR.SBR retrieves the memory address, and optionally the size, of the variable specified as the first parameter. This technique was used in the so called "Speed Optimized Interface" of INFLD under AMOS, and is preserved here just for backward compatibility. (The so called interface is a misnomer under A-Shell, where it actually requires more overhead, and thus is not recommended for new programming.) Getting the address and size of a variable might have some other exotic application, but we are not sure what.

Parameters

var can be any arbitrary variable. *varaddr* can be any numeric type. *varsize*, if specified, must be a 2 byte binary. Note that if *var* is an array element, the size and address will be for that element, not for the entire array. You can get the starting address of the array by specifying the first element, but the only way to get the total size of the array would be to map an unformatted variable on top of the array (with the appropriate size specified in the MAP statement).

GETBYT

```
xcall GETBYT, channel, buffer, bytes
```

GETBYT.SBR reads a specified number of bytes from an open input file.

Parameters

channel (numeric)
is the channel number of the file, which must be open for input.

buffer (unformatted)
Receives the data.

bytes (numeric)
Specifies the number of bytes to read, which must be less than or equal to the size of *buffer*.

Use the EOF(Channel) function to test for end of file condition before each call to GETBYT.SBR.

You can get a similar effect with INPUT RAW, except that it always reads a number of bytes equal to the size of the specified variable. Also see PUTBYT.

GETDEV

```
xcall GETDEV, devstr

xcall GETDEV, unit, dev
```

GETDEV.SBR retrieves the current device information in one of two formats, as shown above.

Parameters

```
MAP1 Devstr,S,6      ! (e.g. "DSK1")
MAP1 Unit,B,2        ! (logical unit #)
MAP1 Dev,S,3         ! (e.g. "DSK")
```

GETJOB

```
xcall GETJOB, jobnam {,progrm}

xcall GETJOB, jobnam {,ppn {,jobnum}}
```

GETJOB.SBR retrieves the jobname, plus optionally the program name or the PPN and job number.

Parameters

```
MAP1 Jobnam,S,6
MAP1 Progrm,S,6
MAP1 PPN,B,2
MAP1 Jobnum,B,<any size 1-5>
```

Comments

Note that the PPN parameter can be mapped in either of the following ways:

```
MAP1 PPN,B,2
```

or

```
MAP1 PPN
   MAP2 P,B,1          ! project
   MAP2 PN,B,1         ! programmer number
```

or

```
MAP1 PPN
   MAP2 PROJ,S,3       ! for example, "001"
   MAP2 PROG,S,3       ! for example, "004"
```

The latter format is preferable, since A-Shell as of build 897 supports decimal PPNs up to 999,999. If you use the B,1 format, you will be limited to octal PPNs update to 377,377.

GETJTB

xcall GETJTB, job-table-variable

There are innumerable subroutines which have been written in order to return the current PPN, terminal name, job name, and so on. Although many of the more common subroutines are implemented in A-Shell, it is clearly not possible to cater to all possibilities. A-Shell therefore also contains this subroutine, **GETJTB** (and its counterpart SETJTB), to return into a structure, everything that is likely to be needed from the job control block, within an application.

The structure of the *job-table-variable* is:

```
MAP1 JOB'TABLE              ! 130+ byte area
    MAP2 JOBNAM,s,6         ! Job name
    MAP2 JOBNO,f,6          ! Entry in job table
    MAP2 JOBDEV,s,5         ! Current device
    MAP2 JOB'FIL1,x,1
    MAP2 PPN                ! Current PPN
        MAP3 P,b,1          ! Project number
        MAP3 PN,b,1         ! Programmer number
    MAP2 JOBPRV,f,6         ! Job privilege word
    MAP2 JOBPRG,s,6         ! Current program name
    MAP2 USRNAM,s,20,""     ! Current user name
    MAP2 TRMNAM,s,6         ! Terminal name
    MAP2 DRVNAM,s,6         ! Terminal driver name
    MAP2 TRMBAUD,f,6        ! Terminal baud rate
    MAP2 JOBDFP,x,6         ! Default file protection
    MAP2 JOBLVL,f,6         ! Class of user (level)
    MAP2 JOBEXP,f,6         ! Experience of user
    MAP2 SYSDOS,s,14        ! Operating system name
    MAP2 JOBATT,s,6         ! Parent job name
    MAP2 SYSVER,s,4         ! Operating sys. version (to 9.9N)
    MAP2 JOBSTS,f,6         ! Job status word
    MAP2 PRJ'STR,s,3        ! Project no. string
    MAP2 PRG'STR,s,3        ! Programmer no. string
    MAP2 JOBTYP,f,6         ! Job type
```

By calling **GETJTB** once at the start of each program, all the information likely to be required is readily available, without the need to call a hotchpotch of routines later on. An implementation of **GETJTB** for running under AMOS is supplied with each copy of A-Shell.

The operating system field, SYSDOS, contains the name of the host operating system on which A-Shell is running, for example **Windows/16** or **AIX**. If A-Shell is running in demonstration mode, then an asterisk is appended to the name, e.g. **AIX***.

The user name field, USRNAM, returns the current logon user name.

> The Jobnam parameter will be returned with the trailing spaces removed, but the Program parameter will be padded with trailing spaces to a length of 6.

The P and PN fields, being only 1 byte each, are only capable of handling up to 377,377 (octal). Since A-Shell supports decimal PPNs update to 999,999, you should use the PRJ'STR and PRG'STR fields instead.

GETMAC

```
xcall GETMAC, addr
```

Where ADDR is either mapped as:

```
MAP1 ADDR
   MAP2 ADDRB(6),B,1
```

Or

```
MAP1 ADDR,S,18
```

GETMAC.SBR returns the hardware MAC ("Media Access Control") address of the Ethernet controller. Depending on how the *addr* parameter is mapped, it either returns it as a set of six decimal numbers in an array, or as a string of the format xx:xx:xx:xx:xx:xx (where each 'xx' is a hex octet). The main purpose of this subroutine would probably be as a security mechanism to prevent unauthorized use of an application, since every Ethernet controller ever produced has a unique MAC address, and there is no practical way to duplicate them. However, before you get too excited about this as a replacement for SSD chips, be aware that there are a number of potential or real pitfalls with MAC addresses. One is that the internal mechanism for retrieving this information varies wildly between operating systems, and thus it is very difficult to guarantee results across all platforms. In fact, as of build 741, the only supported operating systems are Red Hat Linux, AIX, and SCO OpenServer 5.

A second problem with the use of MAC addresses for securing software is that in modern multi-connected, heterogeneous environments, it may not be clear which is "the" Ethernet controller. That is, there may be more than one card in a server, and more than one server in a system.

Third, users may legitimately swap out their Ethernet card for reasons unrelated to software piracy.

> See CONDEV for information on getting the IP address.

GETPPN

```
xcall GETPPN, ppn
```

GETPPN.SBR returns the current PPN in the two-byte format (first byte is project number, second byte is programmer number).

Note that the two-byte PPN format is largely obsolete, as of build 897, which supports decimal PPNs up to 999,999. You are better off using one of the other routines whch retrieve the ppn.

GETPRG

```
xcall GETPRG, prgnam {,sbxnam}
```

GETPRG.SBR returns the name of the currently running program in *prgnam*. If *sbxnam* is specified, it also returns the name of the current routine (if called from within a subroutine). Otherwise, *sbxnam* is set to "". Both parameters are mapped as ten-byte strings. Beginning with A-Shell build 906 of 8 November 2004, **GETPRG.SBR** supports program names up to 10 characters long.

GETUSN

```
xcall GETUSN, username {,condev}
```

GETUSN.SBR returns the current user login name, and optionally the console device name. Both should be mapped as strings of 20 or more bytes.

See OPTIONS=EFFUSR in the MIAMI.INI section of the *A-Shell Setup Guide*, and CONDEV, for related information.

GETVER

```
xcall GETVER, verstring
```

GETVER.SBR returns the version of the currently running program (based on the PROGRAM statement, e.g. “1.0(100)”). This is the same information that DIR/V displays. Map *verstring* as a thirteen-(or more) byte string.

GETX

```
xcall GETX, char
```

GETX.SBR inputs a single character from the keyboard, with function key translation support. The function key translation table will be any PFK loaded into user memory, or the LIB:<tdvnam>.IFX file. The routine will automatically set NOECHO mode (equivalent to `XCALL NOECHO`) and leave the terminal in that mode. It will not display the input character. It is exactly equivalent to XCALL ACCEPN, Char, "IFX".

char may be a floating point variable (in which case the ASCII value is returned), or else a binary or string variable (in which case the raw input character will be returned).

GRECNV

```
xcall GRECNV, dblock
```

GRECNV.SBR takes an input date and converts it to a variety of other formats (string, Gregorian, separated) with optional range and validity checking. The type of conversion is specified by the PCONC field.

```
MAP1 DBLOCK
    MAP2 PSTR,S,12            ! MM/DD/YY or MM/DD/CCYY
    MAP2 PYEAR,B,2            ! Year in Century
    MAP2 PMON,B,1             ! Month
    MAP2 PDAY,B,1             ! Day
    MAP2 PCENT,B,2            ! Century
    MAP2 PBASE,B,4            ! New - Zero for Gregorian date
    MAP2 PJDAT,B,5            ! Gregorian form of the date
    MAP2 PCONC,S,1            ! Conversion request code
                              ! "D" = Start with string (PSTR)
                              ! "G" = Start with Gregorian (PJDAT)
                              ! "M" = Start with separated
                              (PYEAR,PMON,PDAY,PCENT)
                              ! "T" = Start with today
    MAP2 PVALD,S,1            ! To-Gregorian validity checking
                              ! "N" = Do not do reverse conversion check
    MAP2 PFRMT,S,1            ! Date format code
                              ! "C" = MM-DD-YYYY (Yr+Cent*100)
                              ! "S" = MM-DD-YY-CCCC (Sep Century)
    MAP2 PCMPC,S,1            ! Completion code
                              ! "S" = Success
                              ! "R" = Success but out of range
                              ! "U" = Failure
    MAP2 PDOW,B,1             ! Day of week (0=Monday)
    MAP2 POLD,B,5             ! Oldest date within range
    MAP2 PRNGE,S,1            ! Range check code
                              ! "B" = Date must be BC
                              ! "A" = Date must be AD
                              ! "R" = Date must be within old/new range
    MAP2 PNEW,B,5             ! Newest date within range
    MAP2 PFIL,B,5             ! Not used
```

GRECNV was originally written for AMOS by Bob Strunk of Software Systems Consulting in Cincinnati, Ohio, and may be still available if you are looking for a date conversion routine that is available on both AMOS and A-Shell.

GTLANG

```
xcall GTLANG, status, gtlang'map
```

GTLANG.SBR retrieves information from the currently selected language definition file, such as the names of the days of the week, affirmative and negative abbreviations, currency symbol, etc. Using these variables instead of hard coding such information can go a long way towards making your application adaptable to other languages.

Parameters

status (F,6) returns 0 for success or –1 for failure.

```
MAP1 GTLANG'MAP
    MAP2 LANG'NAME1,s,20                MAP2 LANG'USR,x,30
    MAP2 LANG'NAME2,s,20                MAP2 LANG'COL,x,128
    MAP2 LANG'EXTENSION,s,4             MAP2 LANG'JAN,S,20
    MAP2 LANG'CURRENCY'SYM,s,4          MAP2 LANG'FEB,S,20
    MAP2 LANG'CURRENCY'POS,f,6          MAP2 LANG'MAR,S,20
    MAP2 LANG'CURRENCY'SPC,f,6          MAP2 LANG'APR,S,20
    MAP2 LANG'CURRENCY'SIZE,f,6         MAP2 LANG'MAY,S,20
    MAP2 LANG'THOUSANDS,s,1             MAP2 LANG'JUN,S,20
    MAP2 LANG'DECIMAL,s,1               MAP2 LANG'JUL,S,20
    MAP2 LANG'DATE'FORMAT,f,6           MAP2 LANG'AUG,S,20
    MAP2 LANG'TIME'FORMAT,f,6           MAP2 LANG'SEP,S,20
    MAP2 LANG'DATE'SEP,s,1              MAP2 LANG'OCT,S,20
    MAP2 LANG'TIME'SEP,s,1              MAP2 LANG'NOV,S,20
    MAP2 LANG'DATA'SEP,s,2              MAP2 LANG'DCM,S,20
    MAP2 LANG'PPN'LEFT,s,1              MAP2 LANG'MON,S,20
    MAP2 LANG'PPN'RIGHT,s,1             MAP2 LANG'TUE,S,20
    MAP2 LANG'CHARSET,f,6               MAP2 LANG'WED,S,20
    MAP2 LANG'YES'WORD,s,6              MAP2 LANG'THU,S,20
    MAP2 LANG'NO'WORD,s,6               MAP2 LANG'FRI,S,20
    MAP2 LANG'YES'CHAR,s,1              MAP2 LANG'SAT,S,20
    MAP2 LANG'NO'CHAR,s,1               MAP2 LANG'SUN,S,20
    MAP2 LANG'WORD'CHARS,s,30           MAP2 LANG'EXTEND1,B,1
    MAP2 LANG'ULC,s,30                  MAP2 LANG'EXTEND2,B,1
    MAP2 LANG'SPARE,x,30                MAP2 LANG'COL2,X,128
```

`LANG'NAME1` and `LANG'NAME2` are the standard and alternate names for the language.

`LANG'EXTENSION` is the extension used for message files associated with this language. For example, the extension for the ENGLSH.LDF (Language Definition File) is USA, which is why you will find message files with names like ERRMSG.USA, SBRMSG.USA, LITMSG.USA, etc.

`LANG'CURRENCY'SYM` is the currency symbol for this language.

`LANG'CURRENCY'POS` will be set to 0 if the symbol should appear before the currency amount, or 1 if after.

`LANG'CURRENCY'SPC` specifies the “currency spacing” (whatever that is).

`LANG'CURRENCY'SIZE` specifies the number of digits making up the “currency amount” (whatever that means).

LANG'THOUSANDS is the character to be used to separate thousands (i.e. to insert between every three digits). In the USA we use a comma, while in much of Europe they use the period (with the comma being used for the decimal point).

LANG'DECIMAL is the character to be used for the decimal point.

LANG'DATE'FORMAT indicates the date format: 0=month-day-year, 1=day-month-year, 2=year-month-date.

LANG'TIME'FORMAT indicates the time format: 0=12 hour, 1=24 hour

LANG'DATE'SEP indicates the character to be used to separate the day, month, and year parts of a date, e.g. "/" for MM/DD/YY.

LANG'TIME'SEP indicates the character to be used to separate the hours, minutes, and seconds in time (typically ":").

LANG'DATA'SEP indicates the character to be used to separate elements of data.

LANG'PPN'LEFT and LANG'PPN'RIGHT indicates the characters to be used to enclose PPN, (e.g. [7,6]).

LANG'CHARSET indicates the character set number to be used. (This is probably only relevant under AMOS.)

LANG'YES'WORD and LANG'YES'CHAR contain the complete name of the affirmative word (e.g. "yes", "sí", "ouí", etc.) and the standard one-character abbreviation for that word.

LANG'NO'WORD and LANG'NO'CHAR contain the complete name of the negative word (e.g. "no", "non", "nyet", etc.) and the standard one-character abbreviation for that word.

LANG'WORD'CHARS contains all of the characters that can be considered part of a word in this language. (For English, this is just A-Z and a-z, but other languages may allow for accented or other special characters.)

LANG'ULC may specify up to 15 pairs of upper and lower case letters that are to be considered equivalent. The idea here is to identify how to fold accented and other special characters to upper and lower case. (Not necessary for the standard ASCII A-Z.)

LANG'USR has been set aside for user-defined extensions to the language definition.

LANG'COL may be used to specify the first 128 bytes of the collating sequence for this language. It should consist of all the characters 1-128 in ascending sort order. If you are using an 8 bit character set, then the upper 128 bytes may be placed in LANG'COL2 provided that the LANG'EXTEND1 and LANG'EXTEND2 fields are set appropriately.

LANG'JAN thru LANG'DCM specify the full names of the twelve months.

LANG'MON thru LANG'SUN specify the full names of the seven days of the week.

LANG'EXTEND1 and LANG'EXTEND2 must both be set to 90 to indicate that the LANG'COL2 field is valid.

HOST2F

```
xcall HOST2F, alphaflt, hostflt
```

HOST2F.SBR converts an IEEE 4 byte (single precision) or 8 byte (double precision) floating point to the equivalent AlphaBASIC 6 byte format. Although this is done internally by A-Shell all the time, since it uses the IEEE format for internal calculations, the subroutine may be handy when importing data from outside sources.

Parameters

alphaflt
should be mapped as F,6 and will return the converted value.

hostflt
should be mapped as X,4 or X,8 depending on the size of the IEEE floating point value to be converted.

Comments

See F2HOST for the reverse conversion. Also note that when importing data, if you have any control over the formatting of the import file, it will probably be much easier to use string CSV format. See the INPUT CSV statement in the *A-Shell Command Reference* for an easy way to import CSV data.

HOSTEX

```
xcall HOSTEX, cmd {,status}
```

HOSTEX.SBR allows you to execute a host operating system command from within your BASIC program. It is similar to the popular AMOS.SBR known to many AMOS programmers, except that the effect of the command depends on the host operating system.

cmd (string) is a valid command line for the current host operating system, with an optional suffix; see Command Modifiers below.

status (float) is an optional return variable. If specified, it will return the exit value of the command, if any. (Zero generally means success; anything else would be considered an error, but the interpretation of the error is dependant on the command executed.)

Under UNIX, a new process is forked to execute the command as a child process to the current process, but by default the child process uses the same screen console as the parent. If you don't want the output of the child process to disturb the current screen, you can redirect its output to a file (as in the example above). You may also want to redirect the stderr channel to a file as well, as in this example which searches for the string "abcd" in all the .prt files in the current directory, writing its output and any error messages to grep.lst:

```
xcall HOSTEX, "grep abcd *.prt > grep.lst 2>&1"
```

If you want to get the output of the executed command on the current screen, then beware that A-Shell will not know what has been displayed there, and thus the cursor will be out of sync with the display. One way around this is to redirect the output to a file, then read the file back in and use PRINT statements to output it in the context of A-Shell. For example, PWD.LIT uses this technique to print the current working directory:

```
xcall HOSTEX, "pwd > pwd.lst"
Open #1, "pwd.lst", input
Input line #1, PLINE
Print PLINE
Close #1
```

Another technique which might be appropriate for some kinds of interactive commands would be to save the current screen, clear it, execute the new command, then restore the original screen, for example:

```
PRINT TAB(-1,202); TAB(-1,0);      ! save screen & clear it
xcall HOSTEX, CMD$
PRINT TAB(-1,0);TAB(-1,203);       ! clear and then restore it
```

Under A-Shell/Windows, HOSTEX executes both Windows and certain DOS-legacy (also known as "console mode") commands (for which an EXE exists). This will create a new Windows process which will run in its own window (without disturbing the current window, other than perhaps to cover it up).

Command Modifiers

By default, the current program is suspended while waiting for the command to finish executing. (In other words, the command is executed like it was a subroutine.) However, this can be modified by appending a modifier to the end of the *cmd* string, from the table below:

Modifier	Function
<none>	Current process is suspended while waiting for the command to complete
&	Command executes in background (or in the case of Windows, minimized) and in parallel with current job. (Current job not suspended.)
%	(Windows only) Command executes in a minimized window, but current job is suspended while waiting for it to complete. This is most useful for minimizing the screen flash which would otherwise accompany a command that executed very quickly.
$	(Windows only) Command executes in a normal window, but current job is not suspended.

The command modifier, if present, should be separated from the end of the cmd string by a space. For example, to launch the Windows calculator without suspending the current job, you would use:

```
xcall HOSTEX,"CALC.EXE $"
```

Launching A-Shell with HOSTEX

```
xcall HOSTEX,"$ASHELL –e run myprog &"
```

Since A-Shell is itself a valid host operating system command (e.g. "ashell" or "ashw32.exe"), you can use HOSTEX to launch another instance of A-Shell. To make this easier, HOSTEX replaces the symbol "$ASHELL" with the command used to launch the current instance of A-Shell (eliminating the problem of keeping track of how A-Shell was launched, which differs among platforms and installations). The –e switch is needed to force the new instance to exit automatically when the startup command finishes executing. For example, this command would work under either Windows or UNIX, launching another instance of A-Shell using the same executable and miame.ini file as the current session. The new session would run in background (or minimized under Windows) and would automatically terminate when "myprog" ended:

```
xcall HOSTEX,"$ASHELL –e run myprog &"
```

Notes

Under both UNIX and Windows, spawning another copy of A-Shell will not count as another node against the node license. The license banner will be omitted from the spawned copy, but only if both the extension and the path are omitted from the command line, as in the examples. In other words, the directory **C:\VM\MIAME\BIN**, or **/vm/miame/bin**, must be in the environment **PATH**.

If you want to launch a child copy of A-Shell in order to run another BASIC program or LIT commands, the preferred way to do this is with XCALL AMOS, which acts as a front end to **HOSTEX**.

Under Windows, it is possible to execute DOS (or "console) commands with HOSTEX, but only if an EXE for the command actually exists. For example, there is no COPY.EXE (use XCOPY instead). (The console command EXE files are usually stored in the /windows/command subdirectory.) You may also want to check out the Windows START command which is useful for executing other console commands and BAT files. (T Type START /? from the command prompt for help.)

HTLMP

```
command = sbx:HTMLP
```

(Windows only) **HTLMP.**Error! Reference source not found. is a print filter subroutine intended only for use in a COMMAND statement within a printer init file. It creates a temporary copy of the specified print file with an HTML prefix and suffix, and then launches your local browser to view it.

IDTIM

```
xcall IDTIM, stringfmt, idate, itime, flags, status
```

IDTIM.SBR converts a string format date and/or time into so called internal format. (Internal format dates are also referred to as separated format, and is the format returned by the DATE system function. Internal format time is simply the number of seconds since midnight.) The inverse function is ODTIM, which see for converts an converting internal format date and/or time to a string format.

Parameters

stringfmt

should be set to the date and/or time be converted, with the time following the date (with a space separator) if both date and time are included. (Also see *flags* parameter, which indicates if date and/or time are present.) The date, if present, should be in `MM-DD-{CC}YY` or `DD-MM-{CC}YY` format, depending on your language definition file specification for date order. The separator character, however, need not be a dash or even match the language definition file; it can be any non-numeric character. The time format is `HH:MM{:SS} {AM/PM}`. Legal examples would be:

```
12/30/03 13:01
01-01-2004 10:15:33 PM
23:01:10
05.03.1915
```

If only two digits are specified for the year, it is assumed to be in the twentieth century (19xx) unless the SBR=CCYY:## parameter is included in the MIAME.INI, in which case it will determine the YY cutoff below which we assume 20xx instead of 19xx. See the A-Shell Setup Guide for details.

idate (F,6)

will return the internal format date, or zero if the date is not included in the input string. Note, however, that to access the individual separated fields within it, you need to assign the F,6 value to a B,4 which is mapped on top of the separated date structure, as in the following example:

```
MAP1 Sepdate
     MAP2 Mon,B,1               ! Month 1-12
     MAP2 Day,B,1               ! Day 1-31
     MAP2 Yr,B,1                ! Year-1900
     MAP2 Dow,B,1               ! Day of week (0=Mon, 6=Sun)
MAP1 Bdate,B,4,@Sepdate         ! B,4 version of Sepdate, as used by
                                !  the DATE system function
MAP1 Idate,F,6                  ! F,6 version as needed by IDTIM
!
xcall IDTIM, StringFmt, Idate, Itime, Flags, Status
Bdate = Idate
Print "Separated date: ";Mon;Day;Yr;Dow
Bdate = DATE                    ! Get today's date (for comparison)
Print "Today's separated date: ";Mon;Day;Yr;Dow
```

itime (F,6)

will return the internal format time (i.e. seconds since midnight) or zero if the time is not present.

flags (numeric)

should be set to 0 to process both the date and time; 1 if the string is expected to contain only the time, and 2 if it is expected to contain only the date.

status (F,6)
will return a code of 0 to indicate a successful conversion, or –1 to indicate an error (such as an invalid format or invalid date).

IMAGE

```
xcall IMAGE, opcode, handle, status {,parameters...}
```

(Windows only) **IMAGE.SBR** allows you display BMP, PCX, JPG, and TIF format images within the A-Shell window. Note that **IMAGE** requires the installation and licensing of an external graphics library (which resides in VIC32.DLL and VICTW32.DLL)

Parameters

opcode (in, numeric)
Must be set to one of the following:

Value	Function
1	Load image file from disk
2	Close image
3	Display image
4	Open and display image
5	Retrieve information about an image
6	Remove image from display
7	Scan and save image (TWAIN acquire)
8	Select data source
9	Get TWAIN error code

handle (in/out, numeric)
The Comments

Beginning with A-Shell build 936.2.4, XCALL IMAGE (or XCALL AUI,AUI_IMAGE) now supports millirows. Note that as before, and unlike the case with normal controls, the ending row or millirow includes the leading. So, for example, an image with EROW = 4000 will touch an image with SROW = 4000.

Also, removing an image from a dialog now causes the dialog to be repainted immediately. Previously, the image would appear to remain until some other event caused the dialog to be repainted.

Load operation returns this value to identify the image in subsequent calls. It must be passed to the other operations.

status (out, floating point)
is returned from every call, with 0 indicating success. It should be mapped as a floating point, since it can be negative or positive. Possible values of status are given in a table below.

The syntax for **IMAGE** varies with the opcode specified; refer to the following topics for specific information.

Comments

Beginning with A-Shell build 936.2.4, XCALL IMAGE (or XCALL AUI,AUI_IMAGE) now supports millirows. Note that as before, and unlike the case with normal controls, the ending row or millirow includes the leading. So, for example, an image with EROW = 4000 will touch an image with SROW = 4000.

Also, removing an image from a dialog now causes the dialog to be repainted immediately. Previously, the image would appear to remain until some other event caused the dialog to be repainted.

Load

```
xcall IMAGE, 1, handle, status, filename
```

filename is the name of the image file (in either AMOS or native Windows format). Its extension should correspond to the image type. *status* will be returned as zero if success, otherwise an error code. *handle* will be set to an integer value which identifies this image in memory and which must be passed to other calls to indicate which image the call refers to.

Opening the image does not display it. It merely loads it into memory. Although good programming practice would suggest that you close up any open images at the end of your program, as with open files, A-Shell will automatically close them for you at the end of a program.

We have the technology to figure out the image type by looking at the image file itself, but did not see much point in exercising it due to the overhead and the questionable benefit of having images with non-standard extensions. If you see this differently, please contact us to discuss the need.

You can open and hold in memory up to 31 images at a time. This might be useful in some kind of "thumbnail" display application, but keep in mind that the memory requirement for each opened image, regardless of whether or how large it is displayed, is based on the actual image size on disk.

Close

```
xcall IMAGE, 2, handle, status
```

This operation removes the image from the screen and frees up the memory associated with it.

Display

```
xcall IMAGE, 3, handle, status, strow, stcol, erow, ecol, flags
```

This operation displays the image (identified by *handle*) in the position specified by the starting row/col and ending row/col parameters. The image will be automatically scaled to fit in the allotted space, although by default the aspect ratio is preserved. So, for example, if the actual image is square, and you specify coordinates that mark out an elongated rectangle, not all of the space will be used for the image.

flags allows you to specify certain display options, such as whether to stretch the image to fit exactly in the space provided or to preserve its aspect ratio:

Flags	Meaning
IMG_HALFTONE (1)	Halftone print method
IMGF_SCATTER (2)	Scatter print method
IMGF_STRETCH (3)	Stretch image to fit space
IMGF_SCALEQ (4)	Interpolate (hq scaling)

The first two values apply only to printing. If *IMGF_STRETCH* is specified, the aspect ratio is not preserved and the image is stretched as needed to completely fill the specified display rectangle.*IMGF_SCALEQ* forces us to use linear bitwise interpolation (as opposed to simple bit replication) during scaling, which theoretically will achieve a higher quality, less "blocky" image when enlarging beyond the original image size.

The image display will overwrite any screen text that it overlaps, and is not affected by other screen operations such as clearing the screen or saving/restoring the screen. The only way to remove it from the screen is with *opcode* 2 or *opcode* 6.

Images in Dialogs

If a modal dialog is active when the image is displayed, it will be displayed within, and relative to the dialog. If the coordinates given exceed the size of the dialog, the image will be trimmed to the borders of the dialog. This is one good way to display the kinds of horizontal and vertical banner images which are often used in "wizard" dialogs.

Open and Display

```
xcall IMAGE, 4, handle, status, strow, stcol, erow, ecol, flags,
      filename
```

This operation combine *opcodes* 1 (Comments

Beginning with A-Shell build 936.2.4, XCALL IMAGE (or XCALL AUI,AUI_IMAGE) now supports millirows. Note that as before, and unlike the case with normal controls, the ending row or millirow includes the leading. So, for example, an image with EROW = 4000 will touch an image with SROW = 4000.

Also, removing an image from a dialog now causes the dialog to be repainted immediately. Previously, the image would appear to remain until some other event caused the dialog to be repainted.

Load)and 3 (Display) to save you a step. Note that like *opcode* 1, it leaves the image in memory, so you can later redisplay it using just *opcode* 3. See *opcode* 3 for *flags* definitions.

Retrieve Information

```
xcall IMAGE, 5, handle, status, imginfo

MAP1 IMGINFO            ! Image info packet
   MAP2 IMG'OWIDTH,F    ! Orig image width (pixels)
   MAP2 IMG'OLENGTH,F   ! Orig image length (pixels)
   MAP2 IMG'DWIDTH,F    ! Display width (pixels)
```

```
MAP2 IMG'DLENGTH,F    ! Display length (pixels)
MAP2 IMG'BPP,F        ! Bits per pixel
```

This operation retrieves some information about a previously opened image (by specifying the image *handle*). The information is reasonably self-explanatory. Note that the operation can also be used merely to verify is a particular *handle* value is valid.

We may be adding fields to this structure in the future to accommodate additional pieces of information that developers discover a need for. The routine will however always be smart enough to look at the size of the packet you pass so as to not overrun the packet if new fields are added but not yet mapped in existing programs.

Remove from Display

xcall IMAGE, 6, handle, status

This operation is similar to *opcode* 2 except that it leaves the image (still identified by *handle*) in memory so that it can be displayed again later without having to open it. This might be deemed more efficient in some applications than fully closing and reopening the image when switching between various screen displays.

Acquire

xcall IMAGE, 7, handle, status, filespec, comp {,appname}

This operation invokes a TWAIN "Acquire" to retrieve an image from the default TWAIN input device (aka "Data Source"). In the most common case, which involves scanners, this is equivalent to "scan-and-save-image". Manufacturers of image input devices such as scanners typically provide a TWAIN driver and user interface which allows the user to access the functionality of the device. This way, the application does not need to know any details about the device (such as what kinds of controls and options it has). It just knows that once the Acquire operation is started, it will return with a file (if successful) or an error code (if not).

Parameters

handle
: is not actually used here, and is included only for conformity with the other opcodes.

status
: returns 0 for success, else an error code (as with the other *opcodes*).

filespec
: is the specification of the file (AMOS or native format) to store the image in. The extension you specify will determine the image format and must be one of JPG, TIF, BMP or PCX (e.g. TEST.JPG or DSK9:TEST.BMP[100,20]).

comp
: is a compression-related factor determined by the image type. For JPG it ranges from 1 to 100, with 100 being maximum size and quality (minimum compression). 75 is the default and is generally considered the point below which you start to perceive the loss of quality. For TIF it is the compression type (0=none, 2=packbits, 3=Group3, 4=Group4, 5=CCITT Group3). *comp* is ignored for the BMP and PCX formats.

appname

is an optional string containing the name of your application, which, depending on the particular TWAIN driver, may be displayed in the title bar of the device's user interface window.

Comments

Note that the acquisition (scanning) of the image data is independent of the saving of the image file, so it is up to you to specify an image format (via the file extension) that is compatible with the type of scan. (Or conversely, for the scanning operator to select options compatible with the file format.) For example, PCX only supports 1 and 8 bit data, so a 24-bit scan will result in an error while attempting to save the file.

Select Data Source

```
xcall IMAGE, 8, handle, status {,sourcename}
```

This operation allows you to specify which of the available TWAIN data sources (input devices) is to be used for subsequent "Acquire" operations. If *sourcename* is specified and non-blank, it will attempt to select the corresponding data source from among the available options. Otherwise it will pop up a dialog box allowing the user to select from among the available data sources.

This operation (with no *sourcename*) should be exactly equivalent to the "Select Source" menu option that appears on the File menu of most programs that support TWAIN. Refer to the *A-Shell Development Guide* for information on how you can add a "Select Source" option to the A-Shell File menu.

Get TWAIN Error

```
xcall IMAGE, 9, handle, status
```

This *opcode* only applies when the *status* returned from another *opcode* is 11 (TWAIN error). In that case, you can use *opcode* 9 to retrieve a more specific error code, among the following:

Value	Meaning
1	Failure due to unknown causes
2	Not enough memory to perform operation
3	No data source
4	Source already in use
5	Source/Manager error already reported
6	Unknown capability requested
7	Undefined error
8	Undefined error
9	Bad protocol
10	Parameter out of range
11	Message received out of sequence
12	Unknown destination source app
13	Could not create parent window

Value	Meaning
14	TWAIN Source Manager not found

STATUS Codes

The variable *status* is returned from each XCALL IMAGE operation (except for *opcode* 9), with 0 indicating success. Common error codes are:

Value	Meaning
18	Unable to write disk (full?)
17	Unsupported bits per pixel
16	TWAIN function is busy
15	User cancelled scan
14	Could not open TWAIN Data Source
13	Could not open TWAIN Source Manager
12	Could not create TWAIN parent window
11	TWAIN error (See OPCODE 9)
10	Cannot load or link VIC32.DLL (the imaging library module)
9	Misc. XCALL parameter error
8	Error during attempt to scale image
7	Bad image handle
6	Bad OPCODE
5	Unable to allocated memory needed
4	Exceeded max number of open images (31)
3	Unknown image type/extension
2	Image file not found
1	Too few parameters passed to XCALL
0	OK

The remaining errors are returned from within the VIC32.DLL library

Value	Meaning
–1	Range error
–2	Digitizer board not detected
–3	Disk full
–4	Filename not found
–5	Variable out of range
–6	Unreadable TIFF format
–7	8 TIFF bits per sample not supported
–9	Unreadable compression scheme

Value	Meaning
–10	Cannot create file
–11	Unknown file format
–12	Compressed DIB not supported
–14	Insufficient memory for function
–16	Unreadable PCX format
–17	Unreadable GIF format
–18	Print error
–25	Unreadable TGA format
–26	Bits per pixel value not supported
–27	Unreadable BMP format
–34	Function timed out
–40	Could not lock memory
–41	Print function already executing
–42	Invalid image buffer address
–43	Unreadable JPEG format
–44	Image is too complex for operation
–53	LZW compression/decompress not enabled
–68	Handle not valid
–69	TIFF file is in Motorola byte order

INCOM

```
xcall INCOM, name, packet, opcode, stpos, length, sts
```

INCOM.SBR is a variation of COMMON, allowing you to read and write a packet of information in memory, generally for the purpose of passing information between programs run in sequence on one job.

Parameters

```
MAP1 NAME,S,10             ! name.ext of module
MAP1 PACKET (see note)     ! packet to read or write
MAP1 OPCODE,<numeric>      ! 0=read, 1=write
MAP1 STPOS,<numeric>       ! position of first byte in PACKET
MAP1 LENGTH,<numeric>      ! # bytes to read or write
MAP1 STS,<numeric>         ! returns 1=success, else 0
```

Comments

The AMOS version of this routine used a memory module that was loaded into system or user memory. The A-Shell version, on the other hand, actually uses **COMMON.SBR** (which has been enhanced in A-Shell to support many variations), and thus there does not need to be an associated file on disk or even a memory module. The *name* parameter, however, is needed to identify the packet within **COMMON**, but any extension will be ignored.

The main differences between **INCOM.SBR** and **COMMON.SBR** are that **INCOM.SBR** allows you to read/write an arbitrary subset of the packet (rather than the whole packet) and that the INCOM read operation is always non-destructive (whereas **COMMON**'s read is normally destructive but can be overridden by the SBR=COMMONNDR switch in MIAME.INI).

packet may be any type of variable, from 1 to 1024 bytes. You must use the SBR=MSGSIZ:# parameter in MIAME.INI to set the necessary maximum packet length if your *packet* variable will be longer than the default 150 bytes.

INFLD

A-Shell contains a translation of MicroSabio's INFLD.SBR, which was originally written for AMOS and with which it is about 97% compatible. The A-Shell version is actually more sophisticated, and is used internally by A-Shell for the AMOS command mode line editor and even for parts of INMEMO. INFLD is also used as a direct replacement for INPUT.SBR (via the ALIAS=INPUT:INFLD statement in the MIAME.INI file).

Syntax and Parameters

```
xcall INFLD, row, col, xmax, xmin, type, entry, inxctl, v,
      opcode,exitcode, timer, cmdflg, defpt, maxpt, funmap, setdef,
      infclr, hlpidx, maxchrs
```

Parameter	Description
ROW	Row for input
COL	Starting column. Right justified fields will be re-displayed so that they align in the rightmost column (as determined by **COL** and **XMAX**).
XMAX	Maximum size (width; see MAXCHRS)
XMIN	Minimum number of characters to accept. Note that if the **O** (optional) **TYPE** code is used, the minimum is only in effect if at least one printable character is entered.
TYPE	Codes specifying types, formats, etc.
ENTRY	String field contents sent and returned in
INXCTL	INPUT.SBR compatible special exit codes
Parameters above are required; those below are optional.	
V	If zero, multiplies (ROW) by two
OPCODE	Used to specify pre-load, output, other options
EXITCODE	Extended exit codes (how field exited)
TIMER	Used to specify time-out duration
CMDFLG	Controls whether command file input allowed
DEFPT	Specifies default decimal point
MAXPT	Specifies maximum digits to right of decimal
FUNMAP	Controls interpretation of function keys
SETDEF	Defines list or set of allowable inputs
INFCLR	Color specifications
HLPIDX	Various expansion features
MAXCHRS	Max # chars allowed (independent of XMAX)

ROW

ROW specifies the row that the input field is on. Note that if the **V** parameter is missing or evaluates to 0, the **ROW** is multiplied by 2. You can force INFLD into hard-copy mode by setting **ROW** to 0.

You can also specify **ROW** relative to the bottom of the screen (determined dynamically from the terminal driver at runtime) by using a negative number: -1 = the bottom row, -3 = 3rd row up from the bottom, etc.

If you set both **ROW** and **COL** to zero, INFLD will operate in position-relative mode, starting from where the cursor was when you called INFLD. (This is similar to how Basic INPUT works.)

See the discussion on Millirows for more information on specifying and controlling **ROW** variables.

XMAX

XMAX specifies the editing width of the field. Unless **MAXCHRS** is set to another non-zero value, **XMAX** is also equivalent to the maximum number of characters which can be entered. In text mode, the field width will be indicated by marker characters (typically underlines, unless TYPE |**p** or TYPE **i** or TYPE **I** is specified); in GUI mode a Windows-style sunken edit box is used.

Note that minus signs in numeric fields do not count against the maximum number of characters.

ENTRY

ENTRY is the string variable in which the response is returned. If the **OPCODE** parameter is set, then the initial value of **ENTRY** is loaded into the field and used as a default or starting point for modifications. Note that **ENTRY** is returned with trailing blanks unless the **] TYPE** code is used.

INXCTL

INXCTL is traditionally a numeric variable (F,6 or B,2) to accept a return code (as in Alpha's INPUT):

Value	Description
0	Input OK, no message.
1	"Y" answer to YES/NO type field, or Control-e entered to abort record.
2	"N" answer to YES/NO type field, or abort key used (either =Left Arrow or Escape) when enabled with the **E** code)
3	Tab key used to terminate input.

INXCTL is largely superceded by **EXITCODE**, which offers a much wider variety of return code information. Because of this, under A-Shell, you may put **INXCTL** to a different use by mapping it as a one byte string (S,1). In that case, it will return the actual character that caused the field to exit.

TYPE

TYPE is a string containing a sensible combination of the codes listed below. Note that upper/lower case is significant. The codes are organized into categories for easier understanding and itemized below. Note that the codes are generally additive, and you can use as many as you want. For example, specifying both the codes for alphabetic and numeric types will result in an alphanumeric field.

Also note that the codes many be either single-character codes, or double or even triple character codes. All of the double-character codes begin with the lead-in character | (vertical bar.) All of the triple character codes begin with a pair of vertical bars.

Character Type Acceptance

You must have at least one of these. Note that a blank in the first character position of **TYPE** is treated like an **A** to maintain compatibility with an AlphaAccounting quirk.

Value	Description
*	Accept all characters. Same as **A** (see below).
#	Numeric (0-9). If nothing entered, displays a “0” (but returns just a blank string.) Also see **N**.
,	Punctuation allowed. Punctuation is anything other than a number or letter.
@	Control character input.
$	Currency field.
A	All characters allowed (subject to code **b**). Note that **A** is used for compatibility with INPUT, but to most people, **"*"** makes more sense. Note that **A** or **"*"** is equivalent to the combination “**a#,**” (alphabetic, numeric, and punctuation).
a	Alphabetic characters allowed (a-z, A-Z), or as defined by the LDF (language definition file).
C	Carriage Return only
D	Date input.
d	Date input, today.
H	Hours field. Like **$** but omits the “$” from the display.
J	Accepts alphanumeric input, and if the input is wholly numeric, right justifies and fills with leading zeroes.
N	Yes/NO field.
P	Phone number or zip code input.
t	Time input format.
U	European date.
X	Specifies a Y/N field, but with no default. May be used to force the operator to make a choice, or, with XMIN=0, to allow a third option (blank).
Y	YES/No field.
{	SETDEF must specify a string containing all of the allowed characters.

Control character input

Works as a "fast" field to return the first *n* characters typed (where *n* is the value of XMIN) , whether control characters or not. The only character you can't retrieve this way is Control-C. (You can trap that with "ON ERROR GOTO" or by using the **V** code.) Ordinarily you should specify a minimum of 0 or 1 for this to return a single control character properly. Also note that this code over-rides all other codes (where conflicting). See codes **C** and **<**.

Currency field

Redisplays as `"$$###,###,###.##"`. If "**.**" (period) code not used, displays with last 2 digits entered to the right of the decimal point. Otherwise inserts `".00"` after the number entered (if hard decimal not entered) or simply formats the cents as `".#Z"` (if hard decimal entered). Default decimal position and number of digits past the decimal point are adjustable with the **MAXPT** and **DEFPT** parameters. Currency symbol is determined by the applicable Language Definition File (LDF). Note that the symbol may be one or two characters, with or without a space between it and the number, and it may appear before or after the number, depending on the LDF parameters. Also note that by setting **DEFPT** = -2, you can eliminate the decimal point and cents (for handling currency in whole numbers).

Date input

Format will be determined by the combination of **XMAX**, the **u** code, and the language definition file. The possibilities are: **XMAX**=6 (MMDDYY or DDMMYY); XMAX=8 (MMDDYYYY or DDMMYYYY). INFLD forces the entry to be a valid date. See **>** code (for swapping the year up front), **U** (European format), **u** (force language definition file format) and **SETDEF** parameter description (for date range checking) below. See the

GUI-related Codes topic for information on using a Windows-style date-picker control.

Date input, today

If neither **D**, **U**, nor **u** code used, assumes MMDDYY format, otherwise uses format as specified by **D** or **U** codes. See **>** code and SETDEF topic. Note that the pre-load feature only works if the **ENTRY** field is empty.

Yes/NO field

When following a **Y** code, specifies a Yes/No field with "N" default (and pre-display). Otherwise, like **#** but doesn't display a "0" if nothing entered or supplied, and does display one blank after the marker chars. This is a useful alternative to the **#** code when the screen may be cluttered by the display of many fields containing 0 (e.g. `"0"`, `"0.00"`, `"$0"`, etc. (See "B" to display the blanking.)

Phone number or zip code input

If **XMAX** = 7 or 10, specifies phone number and allows alpha and numeric entry. Must be either 0, 7 or 10 characters. Redisplayed as `"(###) ###-####"` with the area code omitted if only 7 digits entered. If **XMAX** = 5 or 9, specifies zip code and allows numeric entry only. Must be either 0, 5 or 9 characters. Redisplayed as `"#####-####"`, with the dash and last 4 digits used only if more than 5 digits entered. Note that this field ignores the language definition file, since there is no guarantee that the phone # or zip being entered actually belongs to the language of the current user. One solution is to prompt for the country first, and then if it is not USA, use alternate **TYPE** codes for zip and phone.

Time input format

(HH:MM:SS {A/PM}). INFLD supplies the separator characters (: for the USA Language Definition File) as needed to simplify data entry, and forces a legal time format (within the limits imposed by the **XMIN** and **XMAX** parameters). See the

GUI-related Codes topic for information on using a Windows-style time-picker control.

European date

Like **D** but forces European format (with the DD digits in front of the MM digits (provided it is not preceded by the **u** code.) The **D** and **U** codes may be reversed for foreign applications by means of a patch. See **>** and **u** codes and **SETDEF** parameter description (below).

YES/No field

Default to (and pre-display as) “Y”, unless followed by a **N** code, or followed exclusively by an **E** or **T** code. (This is to maintain compatibility with INPUT's **YE** and **YT**.) The keyboard character "1" is interpreted as (and converted to) “Y” and "0" as “N" in Yes/No fields. (Note that the English “Y” and “N” abbreviations may be changed by the language definition file (LDF) to more appropriate ones for the local language.)

See the

GUI-related Codes topic for information on converting a Yes/No field to a checkbox.

SETDEF qualifier

Indicates that **SETDEF** contains a list of all the allowable characters. To be accepted, input characters must first pass the normal TYPE code acceptance rules, and then must be listed within the SETDEF string. Consequently, to use **SETDEF** exclusively to specify the allowable characters, combine **{** and **A** TYPE codes. Note that upper and lower case is significant when **SETDEF** is used this way. See the SETDEF topic.

Numeric Formatting Codes

Except for "**-**" and "**.**", the formatting and miscellaneous codes do not affect the characters which can be entered; rather, they only affect the way the characters are formatted. Note that unless otherwise indicated, formatting codes only affect the way a field is displayed on the screen and not how it is returned to the calling program.

Value	Description
. (Period)	Allow decimal point.
- Dash)	Allow minus sign
^ or |u	Force leading minus sign
G	May be useful for aligning decimal points in column of variable precision numbers.
M	Thousands separator
R	Right justify field
v	Value range checking
Z	Zero fill

Allow decimal point

If used with **$** or **H** type and decimal not entered, places decimal to right of last digit entered. For example, if "1234" is entered, it will be redisplayed as "1,234.00". Note that this code affects the action of the OPCODE parameter (see below). When decimal points are allowed, default decimal points are inserted as actual characters into the returned field. If decimal points are not allowed, default decimal points are inserted in the display but not returned.

Allow minus sign

May be typed anywhere in the field, but is always displayed (during entry) to the left of the field. On exit, it is redisplayed as a trailing minus sign (for codes **$**, **H**, **M**), and relocated to the beginning for the returned variable. For example, if "1234-" is entered, it will be redisplayed as "12.34-" (assuming decimal points are not allowed) and returned to the calling program as "-1234."

Force leading minus sign

Force leading minus sign display format when used with **$**, **H** or **M** fields. (Normally these use trailing minus sign display format.) Also note that this only affects the final display to the field; the minus sign is always displayed during field editing at the front, and always returned in the ENTRY parameter in leading format.

Thousands separator

Triggers a numeric thousands separator between every 3 digits to the left of the decimal point in the final redisplay of the field. The actual character used is defined in the language definition table (comma for the USA table). (This is automatic for **$** and **H** fields.)

Right justify field

Note that unlike most formatting codes, right justify (and zero fill) actually affect the way the field is returned as well as the way it is displayed. Additionally, whenever you preload a field using the **R** or **Z** codes, INFLD will automatically strip the leading fill characters prior to the initial display to simply editing.Value range checking

SETDEF parameter must be loaded with a string of the format ",min,max,," where min is the minimum acceptable value and max is the maximum acceptable value. See SETDEF for more info on proper formatting of range strings. Note that you will probably want to use code **s** along with **v** to stifle the "Choices:" message which otherwise appears on the bottom line when SETDEF is non null..

Zero fill

Enough leading zeroes will be added to fill the field to the XMAX size. Note that unlike most INFLD formatting options, the leading zeroes are actually returned in the field. On preloading a **Z** field, the leading zeroes are removed prior to editing, except that if the field is blank, one zero is left.

Other Formatting Codes

The following table shows the formatting codes used for date and alphanumeric data. These do not affect the type of characters allowed, but control the way they are treated upon entry.

Value	Description
=	Do not reformat the field on exit.
^	Convert to upper case
>	Convert date format
c	Capitalize first letter of each word.
|d	Allow date shortcuts
|H	HexaDecade dates (1)
|h	HexaDecade dates (2)
i	(Lower case I) Completely invisible field. Similar to **S** except that nothing is echoed (so you can't even see how many characters are allowed, or how many were typed.)
j	Julian date format
S	Security field: echo all characters as "*". Also see **i** above.
t	Time field
||t	Same as t (time) but in GUI mode returns the time in 12-hour format rather than 24 hour format.

Value	Description
u	Force date editing format to match the current language definition file. Must be specified prior to **d**, **D**, **U** and **>** codes. Note that dates are returned to the program in the same format as they are edited unless the **>** code is use to force MMDDYY storage format.
|u	Convert to upper case (2)

Convert to upper case (1)

Has no effect on non-alphabetic characters. You don't have to specify this for Y/N fields, since they already force upper case. When ^ is used in combination with one of (**$**,**H**,**M**), it takes on a new meaning: forcing the minus signs to be displayed in leading rather than trailing format. (Note that for numeric codes **#** and **N**, this is unnecessary since leading minus display format is the default.) Also see |**u**.

Convert date format

Converts from YYMMDD format (for internal storage) to either MMDDYY or DDMMYY (depending on use of **D**, **U** or **u** codes) for display and input. Note that this conversion will take place on both input and output, so if you want to preload a date field (see OPCODE parameter), it should be supplied in YYMMDD format, and INFLD will also return the date in that format (no matter what the input/display format was.)

Allow date shortcuts

The combination |**d** (vertical bar, lower case d) causes INFLD to allow the use of date shortcuts even if the XMIN value has not yet been satisfied. Otherwise, you must first enter the number of digits required by XMIN before any shortcuts can be used. To review, date shortcuts allow you to enter between 1 and 4 digits and when you hit Return, INFLD fills in the remaining ones based on guessing. For one or two digits, it assumes they are the day of the current month. For three or four, it assumes they are the month and day (or day and month for European format.)

HexaDecade dates (1)

When the combination |**H** (vertical bar, H) is used in conjunction with any Date code, causes the editing format to be automatically expanded from 6 digits to 8 digits (to accommodate the century.) However, on return (and on pre-load) the field is converted to/from 6 digit "HexaDecade" format. In "HexaDecade" format, years past 99 are encoded using one hex digit followed by one decimal digit. For example, year 100 (e.g. 2000) would be "A0", year 105 (e.g. 2005) would be "A5", year 111 (e.g. 2011) would be "B1", etc. The advantage of this system is that it eliminates the need to expand existing date fields from 6 to 8 bytes, yet preserves sort order. It does, however, require that you deal with converting from "HexaDecade" format to "normal" format for printing and other display purposes. (Fortunately, this would typically be a matter of cosmetics rather than processing accuracy, although you may need to also watch out for any mathematical operations on dates.)

Note: Setting OPTIONS=HEXDEC may take care of most or all of the display, printing, and math issues as well. Refer to the *A-Shell User's Guide* for more details.

HexaDecade dates (2)

Using |**h** has the same effect as |**H** but does not expand the field to 8 digits for editing. Instead, it "guesses" at the century. If no other hints given, it will guess 20xx for xx < 60 and 19xx for xx > 60. If you specify |**D** it always assumes 19xx.

Note: You may use SBR=CCYY:## to change the YY cutoff for assuming 19xx vs. 20xx to ## (instead of 60.)

Julian date format

Convert dates to Julian format (for internal storage). May be used in combination with the **D**, **d**, or **u** codes, although if used by itself it will be treated as the combination **Dj**. Note that the ENTRY parameter must be a string of at least 6 bytes which will hold the string representation of the Julian date value; it may then be assigned to a 2 or 3 byte binary number for more compact storage if desired.

Time field

In text mode, field is accepted without editing or formatting, but INFLD will require that the field be in legal time format before exiting.

In GUI mode (|G but not |g), the field becomes a time picker control. Field is returned in military (24 hour) format, unless TYPE ||t specified. XMAX will determine whether the returned format is HH:MM {A/PM} or HH:MM:SS {A/PM}.

Convert to upper case (2)

The combination |**u** (vertical bar, lower case u) provides an alternate way to force the field contents to upper case. The other method, TYPE "^" (caret) causes problems with d/VUE which sometimes wants to treat the caret as a literal control-character lead-in character in your program source code. (This would not only fail to give you the upper case flag, but would also combine with the next TYPE code in your list of TYPE codes to produce a control character which would be, at best, ignored by INFLD.)

Timer-related Codes

These codes allow variations on the field time-out behavior. Note that except for the "**!**" code, all of the remainder have no effect unless the TIMER parameter is specified and is non-zero.

Value	Description

Value	Description
!	Activate return of elapsed time during input. This is only applicable when the TIMER parameter is specified. Note that this can be used even if time-out is not specified (by setting TIMER=0).
+	Disable display of time remaining (on bottom line). Normally, the time remaining until time-out is displayed (and updated every second) on the bottom right corner of the screen.
/	(Forward slash) Disable all timer displays. This differs from the "**+**" code in that the "**+**" code leaves the message "timing out" on the bottom line (but does not show the time)
Q	Disable time-out suspend feature. (Normally, the operator can suspend the time-out counter for 10 minutes by hitting Control-S. Suspending the timer is useful when you have to temporarily interrupt data entry, as when answering the phone.) A-Shell Note: Under A-Shell, CONTROL-B is the suspend key.
r	Time out reminder. This is similar in motivation to the **w** code, except that it just causes a single beep, without any message, after 10 seconds have elapsed (when the timer option is active.)
w	Timer wake up warning. When added to any set of timer related codes, causes a warning beep and message when only 10 seconds remain on the timer. The message blinks TIMING OUT in the lower right corner of the screen, and any key hit during the remaining 10 seconds resets the timer to the original value. Note that **w** overrides **/** to display the warning message and suspend message as well, if invoked. See **r**
|T	*A-Shell (Only) Note:* Overrides TIMER parameter to zero. Useful for disabling timer globally in programs that otherwise use it (by putting this in the SBR=INFDEF: string.)
|t	*A-Shell (Only) Note:* Reset timer to the initial value after every keystroke.

GUI-related Codes

These codes activate various graphic user interface features within the A-Shell/Windows (or ATE) version of INFLD. Note that all of them require GUI mode to be activated with **|G** or **|g**.

Value	Description
||b	For checkboxes, causes the current state of the checkbox to override the normal default state (i.e. the default based on the TYPE codes if OPCODE=0, or the contents of the ENTRY parameter if OPCODE=1 or 2). Useful when the checkbox statement may have been changed by clicking on it outside of the awareness of the application. Same effect as CMDFLG=4.
||C	Causes a field which would normally appear as a combo box or a date picker while editing (see SETDEF, dates) to also appear as a combo box when not actively being edited. Otherwise, these kinds of fields revert to static text controls when not active. See Active Versus Inactive Behavior
||c	Convert Y/N field to checkbox

Value	Description
|E	Causes the field to retain the form of an edit control when not active. (Otherwise, fields normally revert to static text controls when not active.) Also see ||C and Active Versus Inactive Behavior
|F	Fast GUI field
||f	Force fixed-pitch font
|G	Activate GUI enhancements
||G	Same as |G except turns Y/N fields into checkboxes.
|g	Implies |G, but limits the GUI enhancements to just the Windows-style edit box (i.e. without some of the other automatic enhancements such as date picker controls).
||g	Disable/override GUI mode. (Used to temporarily cancel a global |G or ||G or |g)
||H	Disable horizontal scrolling within control. Note that the ability to scroll is sometimes needed in order to fit all of the allowed characters, if using proportional fonts and the field consists of wider-than-normal (e.g. CAPS) characters.
||J	Right Justify Checkbox Text
||K	Select data entry convention
||L	List substitution. SETDEF consists of <code>,<descr> pairs, e.g.: ,01,North,02,South,03,Middle Earth,, Application uses the code items (01, 02, ...) while the operator sees and uses the description items. See Coded Lists
||l	(bar-bar-lower case el) Same as ||L except that ENTRY will return a complete <code>,<description> pair (e.g. "01,North") instead of just the code item.
|M	Multi-line edit box. XMAX sets the display width of the box. MAXCHRS sets the maximum number of characters allowed (up to a maximum of 1024). DEFPT is reinterpreted as the height of the box. Word wrap is automatic. ENTER (along with any other enabled exit key) exits. See Multi-line Edit Control (INFLD).
||M	Variation of multi-line edit. Allows ENTER to be used to manually break lines. Also, displays a vertical scroll bar.
||r	Create a radio button
||S	Variation of combo box in which typing a character only selects the nearest matching item from the list. (Otherwise the user can type independently of the list, and use down-arrow to find the nearest match.) See Combo Box Control (INFLD)
||s	Makes SETDEF list matching optional. Only applies to the combo box, in which case it allows the operator to enter a value that is not in the list.
||]	Prevents field width from being automatically expanded by logic that attempts to allow for the extra space needed by various GUI enhancements. For example, combo boxes smaller than six characters wide typically get an extra three characters of width to account for the space needed for the dropdown button. Similar rules apply to date and time pickers as well.

Active Versus Inactive Behavior

In nearly all cases (text and GUI), INFLD fields assume a different form when active (i.e. being edited) vs. when inactive (i.e. displayed). For example, when active, the data in the field is usually stripped of formatting characters and left justified, while the field itself may be underlined or appear in a different color.

In the GUI mode, these active-vs.-inactive differences may extend to the type of control object used to contain the field. By default, an edit control (or in some cases a date picker, checkbox, combo box, etc.) is used while the field is active, and a static text control is used when the field is not active. This makes it visually obvious which field has the focus, and may discourage users from wanting to move the cursor randomly about the form by clicking. But, it is different from the "typical" Windows form, where there is often no visual change to a field between its active and inactive state, other than the presence of the cursor.

TYPE |E and ||C may be used to achieve an effect more like the just-described "typical" Windows form. |E causes the field to appear as an edit control when inactive. This applies even to combo boxes and date pickers. ||C is similar, but applies only to combo boxes and date pickers (and time pickers), causing them to retain their drop-down button as well. (Neither applies to checkboxes, which always have the same form regardless of whether active or inactive.)

Note that when using either |E or ||C, it is imperative that the field send a mouse-click exitcode string (see HLPIDX) and that your program respond to these exit codes, so that when the user clicks on such a field, the program can retain control. Otherwise, since these control types have their own user interface, the user will have the impression of being able to change the field, but the program will not realize that the change has been made. (In the default case, where the field reverts to a static text control when inactive, this is not a problem, since a static text control doesn't offer any way for the user to directly interact with it, other than simply clicking. You can still define mouse-click exitcode strings and respond to the click by re-activating the field, but the point is that you don't have to.)When in GUI mode, INFLD typically assumes the form of a Windows edit control when editing, and a static text control (with sunken edges) when not currently being edited. For date and time fields, a date or time “picker” (e.g. calendar /clock) control may be used, and for fields with certain kinds of SETDEF lists, a combo box may be used. The codes below may be used to alter this default behavior, and/or to force the use of checkbox and/or radio button controls.

Convert Y/N field to checkbox

(||c) The text message associated with the checkbox must be placed in the SETDEF parameter, and the XMAX parameter should be expanded to allow for the combined width of the text message (prompt or label) and the checkbox itself. Checkboxes may be toggled with the space bar, or set using the same keystrokes as for regular Y/N fields (i.e. either 1/0 for affirmative/negative, or by using the single-character affirmative/negative keys defined in the language definition file). They also return to the program the same field contents as they would have for normal Y/N fields, thus making it quite easy to convert existing Y/N fields to checkboxes.

The color of the text label associated with the checkbox is determined by the OFCLR and OBCLR fields within the INFCLR parameter.

By default, the text label goes on the left and the checkbox itself on the right. This can be reversed with the TYPE **R** (which is normally used for right justification). But in either case, the text itself is left-justified within the space allotted to it. To right justify the text within its space, add SBR=INFLDCBRJ to the miame.ini.

Checkboxes are somewhat unusual, compared to typical INFLD uses, in that a single call combines both the text prompt and the data field. If your programming style divides the functions of displaying the text prompts and the data, you may want to modify the code that previously printed the text label to use INFLD OPCODE 2 to display both the label and checkbox contents at the same time. Having done this, you can leave the XMAX parameter for the checkbox editing call set at 1 (as it would have been for a Y/N field); INFLD will recognize that it overlays an existing checkbox control object and use the existing one.

Fast GUI field

|F generally has the same effect in a GUI field as in a text input field (causing the field to exit automatically when the maximum number of characters are entered). There are some special GUI situations though, where it takes on additional meaning. One occurs with drop-down controls (combo boxes or date pickers), where it causes a single-click on an entry in the drop-down list to cause the item to be selected the the field to exit. (Otherwise, single-click in this case would normally just close up the drop-down list and return to the edit field.)

Force fixed-pitch font

||f forces INFLD to use a fixed-pitch font. The result is the same as if you selected the "Use Fixed Pitch Editing Font" option in the A-Shell/Windows Misc. Settings dialog, except that it only affects this field. Fixed pitch fonts look less Windows-like, but have the advantage of making the relationship between the width of the editing box and the number of characters that can be typed more predictable. (INFLD tries to select the closest available fixed pitch font size that will fit the allowed maximum number of characters.)

Note that while INFLD will use the fixed pitch during editing, it will typically redisplay the field (unless you prevent it with |**R** or overwrite the display yourself) using the current GUI (normally proportional) font. If you don't like that effect (of changing from a fixed pitch for editing to a proportional pitch for display), you might want to consider just changing the GUI font (in the Settings menu) to a fixed pitch font, once and for all. (This will affect all GUI objects though, including buttons, which may look strange using a fixed pitch font.) The other alternative is to use |**R** to stifle the redisplay and redisplay the field yourself using an XCALL MIAMEX 119 function which allows you to specify your own font.

Activate GUI enhancements

|G may be added globally to make INFLD act visually more like a typical Windows control while still maintaining API compatibility with existing applications. In most cases, this simply means that it will use a Windows-style edit box. If the SETDEF parameter contains a list of allowed inputs, this is converted into a combo-box containing those inputs as choices. For dates, it uses a "date-picker" control. For times, it uses a "time-picker" control. See Combo Box Control (INFLD), Date Picker Control (INFLD), Time Picker Control (INFLD), and Checkbox Control for examples and more details about these control types. See ||**c** for checkboxes and |**g**.

Note that by itself, |**G** does not force INFLD to use the standard Windows color scheme (white edit boxes with black text). Instead, it continues to use the INFCLR parameter for setting the color scheme. As a convenience to allow you to quickly experiment with the standard Windows color scheme without tinkering with all of your INFCLR settings, you can use the option in the Misc. Settings dialog to "Force standard colors in edit boxes", which overrides INFCLR.

Right Justify Checkbox Text

||J right justifies text within a check box control. Note that for combo boxes, TYPE **R** determines whether the box is on the right or left of the text, but within the text portion of the control (whichever side of the checkbox), the text would be left justified unless |J is specified. This is equivalent to (but more flexible that) SBR=INFLDCBRJ.

Select data entry convention

||K determines whether to adopt Windows or AMOS conventions related to data entry, when the two are in conflict. For example, without |**K**, the initial input mode is overwrite; with |**K** it is insert. (You can use the INS key to toggle insert mode on and off.) Without |**K**, the HOME key is considered a field exit key (returning EXITCODE 9 if TYPE **9** is specified); with |**K** HOME just moves the cursor to the start of the field.

In most cases where the nature of the GUI version of the control introduces the possibility of navigating within the control using keys that might otherwise have been only used as exit keys (such as up/down arrows within a drop-down list), you can use the CTRL key in conjunction with the navigation key (e.g. arrow, HOME, etc.) to force it to act as an exit key, assuming the exit key is allowed by the TYPE codes.

Create a radio button

(||r) These are similar to checkboxes, but considerably trickier to use because they are must be grouped somehow since only one within a group can be set at one time. If there is only one set of radio buttons on the screen, then these form a single group. Otherwise you must use MIAMEX 119 to create a group box to contain the radio buttons. Note that because of these complexities, it is usually much easier to use a combo box to present a choice of mutually-exclusive options. (Combo boxes are easy to implement – just specify |**G** and use the SETDEF parameter to list the choices.) See Radio button Control.

Miscellaneous Codes

Value	Description
`	(Grave accent) Force destructive enter.
[	Force cursor to blink during input.
]	Strip trailing blanks. Otherwise ENTRY field is filled up with trailing blanks, like in INPUT.
_	(Underscore) Use clear-to-end-of-line function (-1,9) to clear field.
:	Disable bright and dim.
<	Echo a carriage return on exit from the field. This is useful especially with the **F** and **@** codes to let the operator know immediately that the input is being processed.
=	Don't reformat the field on exit. Also see **|R** (don't redisplay on exit).
&	Enable protected fields.

Value	Description
)	(Close parenthesis) Start cursor at end.
\	(Backslash) Clear bottom line.
(	(Open parenthesis) Strip leading blanks from ENTRY before pre-load.
;	(Semicolon) Turn cursor on while waiting for input; turn it off again on exit from field.
0	(Zero) Strip away all characters typed ahead into the input buffer before inputting the field. Useful when you want to make sure that the operator reads a message or prompt before answering it.
B	Disable character blanking.
b	Disallow blanks.
E	Abort allowed.
e	Force non-destructive Return (or Enter). This is the default for most field types except numeric. See code “`” (grave accent) above.
F	Fast input (no Return or Enter needed). Exits as soon as the maximum (XMAX) number of chars entered, or an enabled exit code is used. See code **<**. To alert the operator, INFLD will make the cursor blink while editing fast fields.
g	Return updated field contents.
h	Disables the help message display `"Hit ? for Help"`. This may be desirable when you have help on virtually every field, or when you need to use that area for some other message.
I	(Capital letter I.) Use invisible field markers.
i	Invisible field. Similar to "**I**" but the cursor does not move as characters are typed.
m	Disable the INFLD.DEF option.
n	Return null.
O	(The letter "O") Optional field. Allows operator to skip the field, even if the XMIN specified is non-zero. However, if 1 or more characters are entered, then the minimum goes back into effect.
P	Overrides CMDFLG parameter, setting it to 1 (i.e. forces command file input to be active.)
Q	*A-Shell (Only)* Causes field to be edited in reverse video.
S	Disable the display of the SETDEF options. Normally the list of choices is displayed on the bottom line of the screen.
s	Disable display of SETDEF options, which are otherwise displayed along the bottom line of the screen. Not applicable in GUI mode, where the SETDEF options are loaded into a combo box.
W	Disallow record abort key.
||L	List substitution. SETDEF consists of <code>,<descr> pairs, e.g.: ,01,North,02,South,03,Middle Earth,, Application uses the code items (01, 02, ...) while the operator sees and uses the description items.
||l	(bar-bar-lower case el) Same as ||L except that ENTRY will return a complete <code>,<description> pair (e.g. "01,North") instead of just the code item.
|P*	*A-Shell (Only)* This three-character TYPE combination (vertical bar, P, followed by any character A-Z, [,], ^, _) enables Control+ (whatever the third character of the sequence is) to launch the pop-up utility ASHPOP.RUN.

Value	Description
|p	Prehistoric Compatibility Mode.
|R	*A-Shell (Only)* Don't redisplay field.
|r	Read only
|X*	Program Control-X.
|]	*A-Shell (Only)* Defeats both the padding and stripping of trailing blanks, so that you get back exactly what you typed. (See **]**.)
}	*A-Shell (Only)* Activates "INMEMO" mode. This is a special INFLD mode used internally by the A-Shell implementation of INMEMO.
|}	*A-Shell (Only)* Activates horizontal scrolling mode. Set automatically when MAXCHRS > XMAX.
||}	*A-Shell (Only)* Activates "command-line" mode. This is a special INFLD mode used internally by the A-Shell implementation of the dot prompt and for Basic INPUT statements.

Force destructive enter

This forces a destructive enter ENTER key (e.g. make it always truncate field at position of the cursor.) When using this code, the field is automatically cleared as soon as you enter the first character (except an exit key). Except for numeric fields, ENTER usually exits a field without truncating or erasing any characters. See **e** code (force non-destructive) for information on how INFLD determines whether to use destructive or non-destructive carriage returns.

Disable bright and dim

This disables the output of bright and dim codes. Normally INFLD displays the field markers in dim, with the field contents in bright. On some terminals, use of these intensity commands may cause an inordinate delay, slowing down output. The "**:**" TYPE code will speed up display in this situation by leaving the terminal in the same intensity.

Enable protected fields

Use this code when you have the screen protected. This is necessary since INFLD is forced to disable protected fields in order to display reduced intensity field markers. When it exits, it will enable protected fields if you use this code. (Also note that disabling protection during input is not a problem since it is impossible to move the cursor out of the field, unlike Alpha's INPUT.SBR.) Before using this code you may want to reconsider your use of protected fields. If you are only using them to allow you to clear the data from the screen quickly, then you only need protection enabled when you perform the screen clear function (e.g. `print tab(-1,13); tab(-1,10);tab(-1,14)`)

Start cursor at end

This starts the cursor at end of field instead of the beginning of the field. This is only effective when OPCODE=1 and may be useful in sophisticated applications where a single field is input in parts using multiple INFLD calls. See the parameter description of OPCODE for notes on preloading trailing blanks.

Clear bottom line

This clears the bottom line on exit from the field. This is useful when the program prints error messages on the bottom line. The only time that this would be harmful is when you have a message on the bottom line that should remain there for multiple fields.

Disable character blanking

This disables blanking of characters to right or left of field. INFLD normally displays a few blanks to the right of expanding fields like **$**, **D**, **H** (or to left of field if right justified), in order to completely erase the previous field display. It does not do this for fields which are not re-displayed with extra formatting chars, and you can disable it in all cases with this code.

Disallow blanks

This prevents blanks from being entered into the field. This code also alters the way the Rubout (or Backspace on most PC's) key works. Normally it erases the character to the left of the cursor, leaving a blank if it was in the middle of the field. Since we don't want to allow blanks here, it just acts like a left arrow when in the middle of the field.

Abort allowed

Operator may use the Backspace key (when in first position of field) or Escape key to signal abort. Either key displays and returns the field as "END", and sets both INXCTL and EXITCODE to 2. Note that the Left Arrow key takes on a different function when used without the **E** code; see

EXITCODE. Also note that typing "END" is not supported as an alternative to using Backspace or Escape.

Return updated field contents

Causes INFLD to return the updated field contents when an enabled function key is used to exit the field. Normally the field contents are not returned on a function key exit. (As a mnemonic, you may associate "g" with "GO" commands, where a single key not only exits and updates a field but also causes some other action, dependent on the actual function key used.)

Use invisible field markers

(Usually the maximum size of the field is indicated with underline or period markers.) With this code, there is no visual indication of the field size (although the maximum is still in effect and the bell will sound if you try to go past it. This may be desirable for passwords (in conjunction with **S** code), or for satisfying special aesthetic requirements.

Disable the INFLD.DEF option

The INFLD.DEF option is described more fully in its own section below, but briefly, it allows a method of automatically adding a series of TYPE codes to every INFLD call. **m** will disable this ability, and even disable the search for the INFLD.DEF module, which may shave a few CPU ticks off each call.

Note that INFLD.DEF is replaced by SBR=INFDEF: in the miame.ini file, but the **m** code still serves the same purpose..

Return null

This returns entry as a null string if the field was exited with some sort of abort condition, such as time out, Control-C, Escape, Left Arrow with the **E** code, or Control-E. Otherwise, the pre-existing contents of the field would be returned. This may simplify your abort handling.

Disallow record abort key

Note that with Alpha's INPUT, the record abort key is Control-W (hence the **W** code). However, with INFLD, Control-W is used as an editing function (next word), and Control-E is used as the record abort key. This code **W** should be used whenever you don't plan to check for the INXCTL=1 (or EXITCODE=1) returned code.

Prehistoric Compatibility Mode

The combination |**p** (vertical bar, lower case p) causes INFLD to revert to behavior as similar as possible to the old INPUT.SBR. Some of the differences include: use of dots instead of underlines; always blank 3 spaces to right of field; use Control-W for record abort instead of Control-E; throw away characters above ‘z’ (“{“, “|”, “}”, “~”); and allow months 1-13 and days 1-31 in all dates. This *capability* was added for users converting from INPUT.SBR that wanted to preserve those idiosyncrasies while taking advantage of field editing and other enhancements of INFLD.

Don't redisplay field

This prevents the redisplay of the field on exit from editing mode. This can be useful to eliminate display thrashing when you are going to reformat and redisplay the field yourself manually. In the case of GUI mode, this causes the field control to be deleted on exit (making it easier to redisplay the field using a simple text PRINT statement, but also mandatory that you redisplay it.)

Read only

The combination |**r** (vertical bar, lower case r) prevents the operator from making editing changes to the field. This might be useful in a situation where you wanted to allow the operator to use certain exit codes in the context of a field but not actually change the contents.

Program Control-X

This three-character TYPE combination (vertical bar, X, followed by any third character A-Z) is one of the more obscure features available. It programs Control-X to act like as if you entered Control-*, where "*" is the third character of the three-character TYPE code. For example, the sequence |**X[** would cause Control-X to act like Control-[(ESCAPE). Perhaps this would be useful if your terminal didn't allow certain control sequences to be input from the keyboard.

Exit Key Enabling Codes

These codes are used to selectively enable individual exit keys (and corresponding return codes) you wish to process. For example, if you want to process a Control-R by returning the cursor to home or previous screen, then you will have to enable use of the Control-R. Where possible, the codes are equal to the **EXITCODE** value returned to make them easier to remember. An attempt at exiting a field with a disabled exit key produces a beep.

Note that not all of the available exit keys are enabled in this manner. The Enter key is always enabled (any field may be exited with a Enter). The abort key Control-E is always enabled unless disabled with the **W** code. The time-out abort is activated by specifying a non-zero value for the TIMER parameter.

The following table, like the others, is organized by the alphabetical order of the **TYPE** code characters. See the

EXITCODE parameter for a table ordered by **EXITCODE** values.

Value	Description
1	Enable Escape. Returns EXITCODE=1. Note that this is only effective when used without the **E** code (see above). This differs from Escape with the **E** code in that the contents of the field are not overwritten with "END". Also note that the Control-E abort key also returns EXITCODE=1.
2	Enable Left Arrow. (Backspace on most PC's) This is only effective without the **E** code. Left Arrow returns EXITCODE=2, and is intended to be processed as a move to previous field.
3	Enable Up Arrow Returns EXITCODE=3 and should be processed as either a return to previous field, or to next field above cursor in same column.
4	Enable Control-R. Returns EXITCODE=4 and should be processed as a return to first field in screen, or to previous screen (in multi-screen applications).
5	Enable Control-J (Down Arrow). Returns EXITCODE=5 and should be processed as an advance to next field or next line (skipping fields on current line).

Value	Description
6	Enable Control-T. Returns EXITCODE=6 and should be processed as an exit out of screen (usually to the ANY CHANGE? prompt). This is particularly useful to terminate change mode when the cursor is allowed to be moved about freely between fields (with the above codes).
9	Enable Control-^ (Home). Returns EXITCODE=9 if Control-^ (Home) key hit, and should be processed as a return to first field of screen
7	Enable Spacebar. Returns EXITCODE=13.
8	Enable Rubout or Del). Returns EXITCODE=14 if cursor is in the first position of the field.
?	Help key allowed. Displays message on bottom line "Hit `?' for Help", and returns EXITCODE = 8 if help key entered. Leaves the value of ENTRY as it was before calling INFLD. Note that the help key, message, and message location may be customized.
|a	Enable Control-A (when cursor already at start of field) to return EXITCODE=19. (Control-A normally moves cursor to start of previous word.)
|B	Enable Control-B to return EXITCODE=16, and Control-O to return EXITCODE=17.
k	Enable Control-E to return EXITCODE=15 (rather than it's normal use as the record abort key, which sets INXCTL=1.)
L	Enable Control-L Right Arrow. Returns EXITCODE=12 if right arrow hit when cursor already at right end of field.
|L	The combination **|L** (vertical bar, L) is the same as TYPE **L** (without the vertical bar) in that it triggers EXITCODE 12. The difference is that with **L** by itself, you can only exit from the far right side of the field. (If there are characters to the right of the cursor, then the right arrow key will simply move the cursor to the right.) With **|L**, you can exit the field from any position, provided you haven't already used other editing or cursor motion keys first. This mode is useful for right-arrow to skip from one field to the next.
|N	Enable Control-N to return EXITCODE=22 (instead of just moving the cursor to the end of the field).
|Q	Enable Control-P to return EXITCODE=25 (instead of initiating a screen snapshot).
|S	Enable Control-S to return EXITCODE=26.
T	Allow TAB key as a terminator. Sets INXCTL to 3 and EXITCODE to 7.
|U	Enable Control-U to return EXITCODE=21 (instead of just moving the cursor to the start of the field).
V	Trap Control-C and return EXITCODE=10 (instead of the normal mode which aborts to the BASIC error trapping routine). Note that you must specify the EXITCODE parameter for this to work.
|W	Enable ^W to return EXITCODE=1 and INXCTL=1, instead of moving the cursor to the start of the next word.
|w	Enable ^W to return EXITCODE=20 (when at the start of an empty field)
|XX	Enable Control-X to return EXITCODE=27. Note that this is a special case of the general three-character TYPE sequence **|X*** which causes Control-X to be interpreted as Control-* (where * is any character compatible with the control key).
|Z	Enable Control-Z to return EXITCODE=18 (instead of clearing the field).

V

If **V** is not specified, or contains a 0, the value of **ROW** is multiplied by two. This code is believed to be a holdover from the days when the first 24 line terminals appeared (some sort of VT-model, hence the **V**, replacing the 12 line models.

If **V** is greater than 1, it is interpreted as referencing the group box control whose ID is **V**-1. The current field is treated as a child of that group box, which mainly means that its coordinates are taken as relative to the group box (allowing them to be relocated simply by relocated the group box.

Update Notes: Build 901.1 - 20 Sep 04

INFLD now supports the group box concept better. That is, you can place an INFLD field within a group box, or within a TAB control, by setting INFLD's **V** parameter to the control ID of the parent group box or TAB control (plus 1). (You need to add 1 because the normal value for **V** is 1.)

For example, the following three statements create a group box and a static text control and an INFLD control within it. Note that when a control is specified as belonging to a parent group box or TAB control, the coordinates you specify for child control are offset by the upper left corner of the parent control.

```
! create a group box...(returns GRPID, which must be used to associate subsequent
   controls with this group box)
   xcall MIAMEX, MX'WINCTL, 1, GRPID, "", 0, MBF'GROUPBOX+MBF'LFJUST, "", "",
      STATUS, GRPSROW, GRPSCOL, GRPEROW, GRPECOL, -1, -1

! Create a text label starting in row 2, column 3 of the group box
   ? tab(-10, 20);"1, 0, Input something:, 0, ";MBF'STATIC+MBF'LFJUST;, , , 2, 3,
      2, 20, -1, -1, 0, 0, , , ";GRPID;chr(127);

! Create an INFLD control next to the text label
   xcall INFLD, 2, 21, 10, 0, "A|G", FIELD$, INXCTL, GRPID+1, 1, EXITCODE, TIMER,
      0, -1, -1, -1, "", INFCLR
```

OPCODE

OPCODE, formerly DEFLT, specifies operational options such as pre-load, output vs. input, or format conversion:

Value	Description
0	Normal operation (input). Field is cleared first.
1	Like 0 except contents of ENTRY are preloaded into the field and displayed before input. Note that for codes **d**, **Y**, and **N** (Y/N), ENTRY overrides the standard default only when ENTRY is non-null. Also note that leading fill characters in **R** and **Z** fields, and trailing blanks in all field types will be stripped to simplify editing. You can, however, force it to keep one or more trailing blanks by supplying them as chr$(160) (which is a blank with the 8th bit set).This can be useful in conjunction with the "**)**" code
2	Format and output contents of ENTRY. (No input)
4	Speed and cursor motion optimized version of OPCODE 2; assumes screen already clear.
8	Like OPCODE 0 except returns ENTRY formatted.

Value	Description
9	Like OPCODE 1 except returns ENTRY formatted. Note that you must de-format the field (somehow) between successive OPCODE=9 calls to the same field.
10	Like OPCODE 2, except returns ENTRY formatted.
12	Like OPCODE 4, except returns ENTRY formatted.
28	Format conversion only. Takes ENTRY unformatted and returns it formatted.
33	Read-only input mode. Field loaded with ENTRY, but cannot be changed. EXITCODEs however are still possible. This might be useful in a case were you want to allow a user to review previous entry fields, but only change them by first hitting some kind of special EXITCODE and then entering a password. This has the same effect as TYPE **|r** (vertical bar, r).

EXITCODE

EXITCODE, if specified, receives a value to indicate how the field was exited. Note that for a specific value (except 0) to be returned, it must be activated by a corresponding type code (for the positive **EXITCODE**s), or by the **FUNMAP** parameter controlling function key usage (for the negative **EXITCODE**s).

The number of command or function keys is limited by your terminal and function key translation module, subject to a maximum of 127 for direct function keys, and 999 for "virtual" function keys using the chr(7)+chr(250)+###. sequence.. Note that we use the terminology *Command Keys* (Cmdx) in place of *Function Keys* (Fxx) to underscore the idea that they can be mapped on to the physical function keys any way you like.

Value	Description
0	Return (or Enter)
1	Escape (if neither **E** nor **W** codes used), or Control-W if the **|W** code used.
2	Left Arrow or Escape (when **E** code used)
3	"Up Arrow" (when enabled with **3** code)
4	Control-R (when enabled with **4** code)
5	Control-J Down Arrow (when enabled with **5** code)
6	Control-T when enabled with **6** code)
7	Tab (when enabled with **T** code)
8	(?, HELP) (when enabled with **?** code)
9	Control-^ Home (when enabled with **9** code)
10	Control-C Trap (when enabled with **V** code)
11	Field timed out.
12	Control-L Right Arrow) (when enabled with **L** code)
13	Spacebar (when enabled with **7** code)
14	Rubout or Delete (when enable with **8** code)
15	Control-E (when enabled with the **k** code)
16	Control-B (when enabled with the **|B** code)

Value	Description
17	Control-O (when enabled with the **|O** code)
18	Control-Z (when enabled with the **|Z** code)
19	Control-A (when enabled with the **|a** code)
20	Control-W (when enabled with the **|w** code)
21	Control-U (when enabled with the **|U** code)
22	Control-N (when enabled with the **|N** code)
25	Control-P (when enabled with the **|Q** code)
26	Control-S (when enabled with the **|S** code)
27	Control-X (when enabled with the **|XX** three-character code)
-1	(Cmd1) (When enabled via translation table and/or (FUNMAP))
-2	(Cmd2) etc.
-###	(Virtual function key) Triggered by a function key or a mouse-click on a field which was programmed to send the sequence chr(7)+chr(250)+"###." where ### is any number. See HLPIDX.

TIMER

TIMER, when specified and non-zero, causes the field to time-out (returning **EXITCODE**=11) after the number of seconds specified (in **TIMER**). If the "**!**" code is not specified, then **TIMER** is not modified on exit from INFLD. (This allows you to use a fixed time-out value on every field without having to reset it each time.)

However, if the "**!**" code is specified, then the elapsed time (in seconds) is returned in TIMER. Stand advised that although the elapsed time is quite accurate, the time-out period may become elongated by up to 15% due to heavy system activity. This shouldn't cause any inconvenience, as long as you check for time-out by seeing if **EXITCODE**=11 and not by comparing the elapsed time (returned in **TIMER**) to the original **TIMER** value. For example, if you set TIMER to 30, and the field times out, the elapsed time (returned in **TIMER**) may actually be 29-32 seconds. So check the **EXITCODE** to see if time-out occurred, and if you want to know the precise elapsed time, use the returned value of **TIMER**.

A-Shell (Only) Note: You can override the **TIMER** parameter (force it off) with the **TYPE** code **|T** Also see **TYPE** code **|t** which causes the timeout counter to be reset to the initial value after each keystroke.

CMDFLG

CMDFLG, if 1, causes INFLD to attempt to read from a command file. If no characters are available, it will then accept input from the keyboard. Note that just as with keyboard input, INFLD may reject illegal characters without echoing them. The programmer must take special care when setting up command files to be read in by INFLD. (It may help to set your terminal to a slow baud rate when debugging such command files.)

Note that **CMDFLG** allows you to turn command file input on and off from within the same program. For example, you may have a command file which executes a utility program and supplies the answers to some of the field prompts but allows the operator to enter others.

Note that you can override **CMDFLG** (force it on) using the **TYPE** code **p**.

Setting CMDFLG to 2 (or 3, 6, 7, etc.) when calling a GUI mode INFLD control that contains a drop-down component (e.g. date picker, combo box), will force the drop-down component to be initially displays.

Setting CMDFLG to 4 (or 5,6,7, etc) in conjunction with a checkbox will have the same effect as if TYPE ||b was specified (i.e. forces the default condition to be based on the existing state of the checkbox, rather than on the contents of the ENTRY parameter).

Added in Build 931 of 24 May 2005: option (4) causes checkboxes to ignore the contents of the ENTRY parameter if the checkbox already exists. This simplifies the application logic needed to deal with clicking on checkboxes that generate exitcodes.

DEFPT

DEFPT may be used to specify the default decimal point position relative to the right of the field. A value of zero indicates a default just following the last digit entered. If this parameter is omitted, there will be no default. (If this parameter is passed but you don't want a default, set it to -1.)

The action of **DEFPT** depends on whether the "**.**" (allow decimal point) **TYPE** code is used. If it is, then any default decimal points will actually be inserted into (and returned in) the field. If the "**.**" code is not specified, any default decimal points will be inserted for display purposes only.

Note that the **TYPE** codes **$** and **H** will automatically set **DEFPT** to 2 if "**.**" not specified, and 0 if it is specified. However, you can over-ride this by setting **DEFPT** to anything you want. Also note that you can set **DEFPT** to -2 to edit currency in whole numbers without a decimal point.

MAXPT

MAXPT may be used to specify the number of digits after a decimal point If the "**.**" code is used, then INFLD will prohibit the operator from entering any more than <MAXPT> digits past a hard decimal point. In any case, the display will be padded with enough zeroes or blanks (depending on the **G** code) to have <MAXPT> digits after the decimal point.

If the parameter is omitted, no such checking will take place. However, if you specify the parameter and the value is zero, it will not allow any digits past the decimal point. (If you need to specify the parameter but do not want any such checking, set it to -1.)

Note that the **$** and **H** codes automatically set **MAXPT** to 2 (unless you set it otherwise), and it will always be set to the same as **DEFPT**, if **DEFPT** is specified and **MAXPT** is not specified

FUNMAP

FUNMAP is interpreted as a bitmap controlling the function key translation. If **FUNMAP** is not specified, all function keys will be translated exactly as in the function key translation module. Depending on which of the four function key translation architectures you decide to use (FIXTRN, SET PFK, the INFLD/AMOS

proprietary one, or the A-Shell "virtual" method), **FUNMAP** may be used to override editing translations to return an **EXITCODE** (F#=-#) or select which functions keys are enabled.

A-Shell Note: The only function key translation method which is strictly binary compatible between AMOS and A-Shell is the FIXTRN method. Tables created by FIXTRN.LIT under AMOS and FIXTRN.LIT under A-Shell are binary compatible, assuming equivalent terminal drivers. (The PCTDV and PCTDVG drivers used by A-Shell/Windows are function-key compatible with the AMOS AM65 driver.) SET PFK style translations are operationally compatible between A-Shell and AMOS but the tables themselves are not binary compatible.

When using the FIXTRN or SET PFK function key translation architectures, it is not possible for a single key to have both *editing* and *command* translations, so to get **command** EXITCODEs, you have to set the translation table up to do that directly. The procedure is to translate the function key to Control-G (ASCII value 7) followed by the ASCII character whose value is the EXITCODE (negative) you want. For example, you might translate **F5** to Control-G Control-E. Since the ASCII value of Control-E is 5, INFLD will translate this sequence to EXITCODE -5. (Control-G was chosen as the lead-in since it is generally reserved for some non-editing auxiliary function in other editing utilities. Prior to version 6.1 however, INFLD used Control-B for this function.)

Since you have to translate all function keys directly to EXITCODEs in the translation table when using FIXTRN or SET PFK to make your tables, the FUNMAP parameter only controls which function keys will have command translations enabled. The standard plan would be to set up the .IFX module to translate all the F keys (that could ever be used for command EXITCODEs) as discussed in the preceding paragraph. Then the FUNMAP parameter is set at runtime to enable selected function keys, according to the table above. For example, to enable F1, F3, and F5, you would set FUNMAP = 1 + 4 + 16. The main difference here between using the proprietary translation architecture module and either of the SET PFK or FIXTRN architectures is that with the proprietary format, merely setting the appropriate bit in the (FUNMAP) parameter will enable the function key, while with the SET PFK/FIXTRN formats, you have to both set the appropriate bit in FUNMAP and define the appropriate translation within the .IFX module.

Since FUNMAP contains only 32 bits, it can only enable/disable F1 thru F32 directly. Any function key value beyond F32 is enabled by the high bit of **FUNMAP** (2**31).

There are two other possibilities for command function key EXITCODEs. One is to key them in directly from the keyboard at run time. From a programming standpoint, the **FUNMAP** parameter is used exactly the same with the alternate keyboard sequences as with the SET PFK or FIXTRN translations.

The other is to use the new A-Shell "virtual" function key codes, which are typically associated with clicking the mouse on a field or button. These are defined using the sequence chr(7) + chr(250) + "###." where ### may be any number, 1 thru 99999. This method allows you associate a (negative) **EXITCODE** value with just about every object or command in your entire application. See the **HLPIDX** parameter for associating such sequences with inactive INFLD objects, and XCALL MIAMEX 119 for associating them with other kinds of control objects (buttons, static text fields, etc.)

SETDEF

SETDEF is a string specifying a set or list of allowable inputs, and can be used in one the following ways.

List of Complete Inputs

If the field is not a date, and the **TYPE {** is not specified, then **SETDEF** is interpreted as a list of possible complete entries. When specified, the choices will display on the bottom line of the terminal (as many as will fit) to assist the operator (unless disabled with the **s** code). A null string disables the feature (accepts all inputs)

The format of the string is:

```
/x1/x2/x3/x4/..../xn//)
```

Where:

`/` Represents the element delimiter. It may be any non-numeric character except the minus sign. We suggest slash or comma. Note that the delimiter character must appear once in the first position, once between each element in the list (all blanks are significant), and twice at the end of the list

`xn` - The elements x1, x2, etc., represent the allowable inputs. They may be any length (up to the length of the field) and may differ in length. All characters are significant, as is upper/lower case.

The list elements may contain any of the following wildcards:

Wildcard	Result
*	Matches anything. Used only at the end of an element.
?	Matches any one character.
#	Matches any numeric digit.
@	Matches any alphabetic digit.

A-Shell (Only): In GUI mode (see GUI-related **TYPE** code **|G**), a field with this type of **SETDEF** list will be converted to a drop-down combo box. The user can select an item from the list using the mouse, or type freely as before (provided the results match one of the entries in the list). Note that in this mode, the length of the items in the list may be longer than **XMAX**, in which case the entry only has to match the first **XMAX** characters of an item in the list (and only the first **XMAX** characters are returned to the calling program). An example of a situation where this would be useful is a list of states, where you are only interested in the 2 character code, but to be helpful to the operator, the dropdown list could also contain the full name of each state (e.g. "CA California, WV West Virginia," etc.)

> See **TYPE ||C** for forcing the field to be display as a combo box even when inactive. Without **||C**, after exiting from the combo box editing mode, the field will be redisplayed as a normal sunken, static text field. Also see **||S** and **||s** for variations on the combo box matching logic.

Dummy List of Inputs

In GUI mode, if **SETDEF** is set to three dots ("..."), then INFLD appears as combo box, but any action that would cause the drop-down box to appear (clicking on the drop down button or hitting ALT-DOWN) instead causes INFLD to exit with **EXITCODE** 29. The idea is to to allow the application to then build its own list of choices based on the partial data entered in the field. See Self Service Combo Box

List of Paired Inputs

If TYPE ||**L** or ||**l** specified, then SETDEF is interpreted as list of pairs, where the first item in each pair is the abbreviated form and the second member is the descriptive form, i.e.:

```
/01/South/02/North/03/Middle Earth/04/Purgatory//
```

In the list above, 01,02,03 and 04 are the abbreviated forms what will be used by the application internally, whereas "South", "North", etc. are the descriptive forms that the user will user and type (or select from the combo box.) See the TYPE code definitions for more details.

Date or Value Range Checking

For numeric fields, when the **TYPE v** is specified, the **SETDEF** parameter must contain a two-element list consisting of the minimum allowed value and the maximum allowed value. For example, ",23,99,," would allow a range from 23 to 99. Note that the delimiter used to separate the two values must be a valid AMOS numeric delimiter, such as space, comma, or backslash.

For date fields, **TYPE v** is not specified, and the **SETDEF** parameter may be used to specify the first and last allowable dates. To use this feature, specify a standard format **SETDEF** list with only two elements: the beginning and ending dates. The dates must be in YYMMDD (XMAX=6) or YYYYMMDD (XMAX=8) format (regardless of the input format).

Examples:

SETDEF string	Result
/M/F//)	Matches only `M' or `F'
,TALL,SHORT,PORTLY, PETITE,,)	Matches `TALL', `SHORT', `PORTLY', or `PETITE'.
|##-*|@@#||)	Matches anything starting with 2 numeric digits followed by a dash, or any 3 character field in which the first two are alphabetic and the third numeric.
\M@*\\)	least two alphabetic characters, with the first one `M'. (MR., MRS., MS., etc.)
\850101\851231\\)	When used with a date TYPE code (**D**, **d**, or **U**), will match any date between 1-Jan-85 and 31-Dec-85, inclusive.
,-40,49,,	When used with TYPE **v** in a numeric field, will match only values ranging from negative 40 to positive 49.

Character Acceptance List

The third way of using **SETDEF** is to specify a list of acceptable characters (rather than complete field entries). To use this method, specify the "**{**" code and then load **SETDEF** with a string of the characters you want to accept. Note that any input characters must first pass the normal **TYPE** code test and then must be in the **SETDEF** string, so you may wish to specify the **A** TYPE code as well. For example:

```
SETDEF = " 123ABCabc+-"
```

This would limit input to the numbers 1, 2, and 3; the letters A, B, C, a, b, and c; the plus sign, minus sign, and space. Note that upper and lower case must be listed independently.

INFCLR

INFCLR allows the specification of various color palettes which are then automatically invoked for the corresponding situations. If specified, it must be mapped as in the file INFPAR.BSI (provided with the INFLD release.)

The effect of any of these colors can be disabled by setting it to -1, which causes INFLD to use the color setting which was in effect when INFLD was called. The standard AM72 color selections range from 0-15, where 0 is usually black, so if your field seems to disappear, you should check if you accidentally left any of the color palettes 0 on 0 (black on black).

Color	Codes	Description
Display	DFCLR, DBCLR	Used when INFLD is called to display a field (OPCODES 2, 4, 10 and 12). The editing colors (EFCLR and EBCLR) are used while you are editing a field (unless it is a negative number).
Negative	NFCLR, NBCLR	Override the editing palette whenever the field contains a negative number. The color change occurs instantly whenever you enter the minus sign or plus sign keys to change a number's sign.
Negative fore-ground	NFCLR	Also overrides the display and update foreground colors (DFCLR and UFCLR) whenever the field is negative. Typically, NFCLR will be red (#4 on the standard AM72 palette).
Update	UFCLR, UBCLR	Used to redisplay the field after editing, regardless of whether the field was changed. The intent of this feature is to indicate which fields have been edited on a "CHANGE" screen; we recommend a color very similar to the display color.
Message	MFCLR, MBCLR	Used for displaying any auxiliary messages, like time out, invalid entry, help available, etc.
Original	OFCLR, OBCLR	Used when clearing any messages. You should set them to match the normal color scheme of the area where messages will be displayed. *A-Shell Only:* these are also used to specify the color scheme for the text portion of a checkbox or radio box. See TYPEs “||c” and “”||r” in the GUI-related TYPE code section.
Forms	FFCLR, FBCLR	Used to display the literal characters embedded in a form string (see TYPE “f”.)

INFLD will automatically return to the pre-existing color scheme on exiting from a field, so you don't need to worry about it changing your ambient colors.

INFLD does not check whether your terminal supports color, relying instead on your terminal driver to discard unsupported commands. You should program as if color was supported, since in most cases, color will have no effect on monochrome screens (unlike the PC where color commands may be interpreted as other video effects on monochrome monitors). However, by passing all color commands to the terminal driver, we leave you the option of modifying the driver to substitute color commands for some other attribute. (Should you attempt this, note that the color commands should not occupy a space, and operate as mode attributes.)

A-Shell (Only) Note: In GUI mode, there is a convenient option in the Misc. Settings dialog which overrides **INFCLR** to force INFLD to use the standard Windows color scheme (typically black text within a white edit box).

> The INFCLR parameters will be ignored in both of the following situations: 1) When the "Force standard Windows colors in edit boxes" option is checked (in the Misc. Settings Dialog); 2) When XP Themes are active.

As a convenience to those not interested in infclr but who want to use a subsequent parameter, the *infclr* parameter (normally a mapped structure) may be reduced to numeric zero. Thus, for example, you can call INFLD to input an alphanumeric scrolling field with an editing window width of 60 characters and a maximum width of 150 characters as follows:

```
xcall INFLD, row, col, 60, 0, "a", entry, inxctl, 1, 0, exitcode, timer, 0, -1, -1,
    0, "", 0, "", 150
```

HLPIDX

HLPIDX is ignored under AMOS, and may be used for any combination of tool tips, mouse click strings and help keys.

Tool tip

In GUI mode, you may specify a “tool tip” (a short text message which appears automatically when the mouse idles over the field position for a second or two). These are handy for giving the user additional prompt information that would otherwise clutter up the screen. The format is simply a “>” followed by the text you want to appear.

Mouse Click String

In order to implement the effect of allowing a user to be able to click on a field in order to edit it (as is typical in Windows forms), you need some way of telling your application that the click event has happened. The recommended way to do this is to encode unique high-numbered pseudo-function-key exit codes with each field. The format for this is:

```
Chr(7) + chr(250) + "###."
```

In the above example, the ### characters should be replaced with the number of the pseudo-function key, for example, “231.” for function key 231. Since function keys normally set **EXITCODE** to the negative of the function key number, this would cause the current INFLD operation to exit with **EXITCODE** -231.

To further clarify with an example, lets say you have a screen with 20 fields, each of which uses the above technique to associate itself with a pseudo-function key 201 thru 220. If the user is currently editing field #1 and then she clicks on field #15, the INFLD call to edit field 1 will return **EXITCODE** -215, allowing your program to then do any post-field processing you desire before jumping to the routine to edit field #15. Obviously this does involve some programming effort, but if systematic logic is used for field editing, it is usually not too hard to add this kind of logic to an existing program.

If **HLPIDX** contains both a tool tip and a mouse click string, then the tool tip must be in the first position, followed by a chr(126) delimiter character, and then the mouse click string, e.g.:

```
HLPIDX=">Tool tip first" + chr(126) + Chr(7) + chr(250) + "213."
```

> The sample syntax for the mouse click string above suggested a three-digit number, but in fact, they can range from one to six digits, thus allowing you, if desired, to assign a unique code to every field in your entire application.

Help Key

The original intended use for **HLPIDX** was to store a string to be used as key to link to some information in a help database. Sadly, this has yet to implemented as originally envisioned (i.e. as something INFLD would respond to without the application being aware of it), but the possibility remains. And until then, many applications may find it useful to use the parameter as envisioned in order to implement their own help logic in a routine which serves as a wrapper to INFLD.SBR.

If **HLPDIX** contains a traditional help key in addition to a tool tip and/or mouse click string, then the help key string must be first, e.g.:

```
HLPIDX="Traditional help key>Tool tip next" + chr(126) + Chr(7) + chr(250) + "213."
```

MAXCHRS

A-Shell (Only). May be used to separate the display width of the field from the maximum number of characters that can be entered. Normally the objective here is to allow a longer string to be input than screen real estate would otherwise allow. This is accomplished by scrolling the field horizontally within the display width defined by **XMAX**. A secondary motivation presents itself in the GUI proportional font environment, where you might want to make **XMAX** larger than **MAXCHRS** in order to allow for the possibility that all the characters entered are wider than average, and thus wouldn't fit within the display box size based on **XMAX**.

For example, a two-character state field is likely to have this problem. The field display width is calculated based on the fixed pitch font size times the **XMAX** value. In most cases, this is more than wide enough for proportional fonts, which "on average" are more compact than fixed pitch fonts of the same point size. But capital letters, particularly in a combination like "WY", are likely to require more space. In this case, setting **MAXCHRS** to 2 and **XMAX** to 3 might be a good idea.

If **MAXCHRS** is omitted or equals zero, it is treated as being equal to **XMAX**.

Horizontal scrolling: The parameter *maxchrs*, which is only supported in the A-Shell version of INFLD, may be used to specify a maximum horizontal scrolling width. If specified, and if larger than *xmax* (the field editing width) then the field will support up to *maxchrs* characters, scrolling horizontally as required within the display window defined by *row, col* and *xmax*.

As a convenience, especially to those interested in using the horizontal scrolling feature but who do not care about color, the *infclr* parameter (normally a mapped structure) may be reduced to numeric zero. Thus, for

example, you can call INFLD to input an alphanumeric scrolling field with an editing window width of 60 characters and a maximum width of 150 characters as follows:

```
xcall INFLD, row, col, 60, 0, "a", entry, inxctl, 1, 0, exitcode,  timer, 0, -1, -
   1, 0, "", 0, "", 150
```

External Parameters

In addition to the calling parameters, there are several external parameters which affect the operation of INFLD.

Parameter	Description
Language Definition File	INFLD gets a number of format and character symbol options from the Language Definition File (LDF) for your job. If no LDF file is in effect (on older versions of AMOS), the USA defaults are used.
Set TYPE code	*AMOS only.* INFLD Definition File.
Set TYPE code	*A-Shell only.* This is the A-Shell equivalent to INFLD.DEF. The command **SBR=INFDEF:<TYPE code string>** is placed in the A-Shell configuration file (miame.ini) and processed when A-Shell starts up. After that, the specified TYPE code string is added to the end of the TYPE parameter each time INFLD is called. (except when the **m** code is used.)
Century Definition	*A-Shell only.* The miame.ini parameter **SBR=CCYY:##** may be used to change the cutoff for which years are assumed to be part of the current 19xx century. Any year greater than ## will be assumed to be 19##, while any year equal or less than ## will be assumed to be 20## (unless overridden by some other parameters). Note that this applies to ODTIM and IDTIM as well as INFLD.
SBR=	*A-Shell only.* Because many dealers have modified their version of INFLD and want their modifications to be standard in A-Shell, we have added a number of other option switches that can be installed into A-Shell using the **SBR=** parameter in the miame.ini file. Consult the *A-Shell User's Guide* and UPDATE.TXT files for more information on the available options.

INFLD Definition File

INFLD looks for INFLD.DEF in system or user memory at runtime. If found, it reads the first line of characters from the file and interprets them as additional TYPE codes. This is a handy way of experimenting with many of the miscellaneous and TIMER-related TYPE codes which you could safely add to any category of field.

To create the INFLD.DEF file, just use VUE or another editor to create an ordinary text file whose top line is a string of valid TYPE characters. Save the file to disk and use the LOAD or SYSTEM commands to load it into user or system memory.

Searching for the INFLD.DEF file adds another few jiffies of execution time to INFLD. You can reduce this by loading INFLD.DEF into memory immediately after the INFLD.SBR. (INFLD checks there first before doing a general search.) If you are not using the INFLD.DEF facility, you can disable the search for it in memory by using the "**m**" TYPE code.

Comments and Updates

Following are miscellaneous comments about INFLD usage, and update notes from recent changes.

Screen snapshots

Most of the differences between the AMOS and A-Shell versions are only relevant to system-level programs, but there are some differences that may affect ordinary users. One is that the Control-P command, under the A-Shell version of INFLD, brings up a screen snapshot utility. The utility gives you the option of printing a copy of the screen, and/or just saving it in the file <jobname>.BUF. If saved, subsequent pictures will be appended to this file, creating a snapshot log which could be useful in a variety of situations (documentation, error reporting, etc).

> The name of the screen snapshot capture file can be changed using the **–b** and **–ba** A-Shell command line switches.

Pop-up Utilities

Another unusual feature implemented only in the A-Shell version of INFLD is the pop-up utility command. This command will launch a new session directly on top of the existing one and start it running BAS:ASHPOP.RUN, which is intended to be a small pop-up utility (or menu of utilities). The command itself is defined and activated by means of the *type* code string `|Px` (where **x** is the command letter to be combined with the Control key). For example, if *type* contained `|P]` then Control-] would launch the ASHPOP utility. A-Shell is released with a sample ASHPOP that displays a simple menu of two choices, one of which then refers to another utility, ASHCAL, a simple calculator. The source for both can be found in DSK0:[7,376] As an added convenience, the calculator will write the accumulated result to a file `MEM0:<jobnam>.CLP[1,1]` if you exit with TAB, and INFLD will pick up the results from that file and load it into the current field. This of course requires that the disk device MEM0: be defined and that a ppn [1,1] exists. If you want this feature available all the time, then add the TYPE code to the default string in MIAME.INI (e.g. SBR=INFDEF:|P]. You may want define a control key other than Control-] though, since Control-] is the terminator key for some telnet utilities.

A number of differences between the two versions can be adjusted by means of SBR statements in the MIAME.INI file; see the *A-Shell Setup Guide* for more details.

> If your application was previously using INPUT.SBR, and you want to preserve all of the idiosyncrasies of INPUT.SBR, then use SBR=INFDEF:|p in the MIAME.INI file to invoke "prehistoric compatibility mode."

Update: Build 861 - 25 Jan 2004

(UNIX) Support SBR=INFLD_KEEPALIVE to send a harmless byte sequence every 15 seconds while waiting for input within INFLD (which includes the dot prompt and BASIC INPUT statements). This serves two purposes. First, it will prevent connections from being dropped due to lack of activity while they are actually in a program waiting for input. Second (and perhaps more important), it provides a mechanism for UNIX/Linux servers to detect connections that have been dropped (without having to wait for the standard keepalive timers to expire, which by default may run to 2 hours). This way, within 15 seconds of the connection being dropped, the server will send a packet, which cannot be delivered. This should trigger a retry timer which will try to resend the packet some number of times before concluding that the connection is dead. This process is generally much quicker than the keepalive detections process.

Update: Build 862 - 02 Feb 2004

When mouse cursor reporting is active, INFLD now treats a mouse click outside of the field as an exit condition, returning EXITCODE -47. You can then retrieve further information about the last click by calling MIAMEX 125 (see next).

Update: Build 875.3 - 14 Mar 2004

TYPE i now silences the beveling so that the field remains totally invisible (no cursor, no indication of field position or size, no echoing of characters).

Update: Build 877 - 01 Apr 2004

(Windows) INFLD TYPE |G now invokes a Windows-style edit control implementation of INFLD. Most of the standard INFLD features continue to work, although from the user's standpoint, it looks and acts like a typical Windows edit control. You can add this feature globally by adding |G to your SBR=INFDEF: entry in the miame.ini file. Note that there are some subtle differences between the edit control implementation of INFLD and the standard text implementation. One of the most noticeable is that since single character edit controls do not work very well, we automatically expand the size of any single character field to a width of two. We don't allow two characters to be entered, but the extra space might pose a slight screen layout problem in extremely crowded screens.

Many of the INFLD error messages now appear as either message boxes or at least using proportional fonts. (Note, this also applies to UNIX as long as the client is ATE.) The various timer-related displays have been simplified down to a simple display of the time remaining. Clicking on the clock will reset it to 10 minutes (provided suspend has not been disallowed.)

The HLPIDX parameter of INFLD may now contain the keyboard string to be sent if the field is clicked on when it is inactive. The format is HLPIDX = <optional other help string> + chr(126) + <kbd click str>. This only applies when TYPE |G is specified and TYPE = (disable field redisplay) is not specified. The idea here is that since INFLD will automatically turn the edit field into a static text display control on exit, if you want to implement the ability to click on a field to edit it, then you can specify the appropriate keyboard command string when you call the field to be edited.

Prior to edit 883, the lead-in was chr(127); from edit 883 on, the lead-in is chr(126)*

The INXCTL parameter to INFLD may now be specified as a one byte string or binary, in which case it returns the actual character used to exit the field. This is more useful to many people than the rather limited information provided by the traditional INXCTL.

Update: Build 881 - 14 Apr 2004

INFLD supports a new mechanism for encoding pseudo-function keys which works better in heterogeneous environments involving ATE or the Windows control implementation of INFLD. The new scheme also allows for an unlimited range of virtual function keys (whereas the previous scheme only really supported F1 thru F127.) The new encoding format is:

```
CHR(7) + CHR(250) + FNUM$ + "."
```

where FNUM$ is the string representation of the function key number (e.g. "1" for F1, "793" for F793). Since the number has a variable length, a terminator character is required. In the above example we use a period, but any non-numeric character will work.

As an example, if you want to set up an INFLD field so that when you click on it, it acts like a virtual F122 (i.e. returning EXITCODE -122), then you could set the HLPIDX string as follows:

```
HLPIDX = chr(127) + chr(7) + chr(250) + "122."
```

As of Build 883, this syntax as been changed to:

```
HLPIDX = chr(126) + chr(7) + chr(250) + "122."
```

Note that this takes advantage of a new INFLD feature relating to HLPIDX (which see below in this document). The leading CHR(127) is needed to tell INFLD that what follows is a mouse click string.

Update: Build 882 - 18 Apr 2004

In INFLD, when INXCTL is a one byte string and the use transmits the new pseudo function key sequence (see item 2 under edit 881 below), INXCTL will return 250. (This allows these sequences to be distinguished from the normal IFX function sequences, which set INXCTL to 7.) This distinction is useful when the new pseudo-funkey sequences are used to encode mouse clicks, thus avoiding the need to start the mouse pseudo-funkey codes at some offset to avoid confusion over F-key generated function codes.

The INFLD edit control (|G) now uses the TYPE sequence |K to determine whether to adopt Windows-style or AMOS-style conventions whenever the two are in conflict. For example, without |K, the initial input mode is overwrite; with |K it is insert. (You can now use the INS key to toggle between insert and overwrite mode.) Without |K, the HOME key is considered a field exit key (returning EXITCODE 9 if TYPE 9 is specified); with |K, HOME just moves the cursor to the start of the field (but you can use CTRL-HOME to get the exitcode effect.

Update: Build 883 - 21 Apr 2004

(ATE) INFLD on a server with ATE on the client now executes locally on the client when TYPE |G specified. There are some issues yet to be worked out, so don't expect to put this into production. (But do send us bug reports.) Requires edit 883 or higher on both server and client.

Three known problems are:

- The cursor will be turned off, so you may need to turn it back on manually when you exit the program.
- Control-C generally doesn't work, unless you use TYPE V. (To overcome this, we need to implement a means for ATE and the server to synchronize their SET CTRLC/NOCTRLC settings.
- The control is noticeably slower on some platforms to bring up than the standard INFLD. It does not appear to be much of an issue with Linux.

HLPIDX syntax (see item 2 under edit 881 below) has been changed to use chr(126) as the lead-in instead of chr(127). **Warning: you must correct any existing programs using this technique or they won't work properly under this and later releases.** The choice of chr(127) was a bad one because it conflicts with the remote/ATE implementation of INFLD (which works through the TAB(-10,x) interface and thus uses chr(127) as a terminator. Hopefully few programs will be affected.

Update: Build 885 - 03 May 2004

INFLD now uses a date-time "picker" control for date (TYPE D) and time (TYPE t) fields when in GUI (TYPE |G) mode. Note the following idiosyncrasies of this control:

- The date picker takes up more space than the traditional MMDDYY (or DDMMYY) format of date entry, so the control is automatically elongated by a few extra columns. Although this might possibly overwrite something to the right of the control, the effect is only temporary, since as soon as you exit from the control, it is redisplayed using the normal size and format.
- If you set XMIN to 0, the date picker control will include a checkbox which the user can uncheck to indicate no date. Otherwise there is no way to enter a blank (or invalid) date.
- The left and right arrow keys will move between the sub-fields of the date or time. If you want to pass these arrow keys to INFLD (to be treated as possible field exit keys), you'll have to hold down the SHIFT or CTRL buttons while hitting the left/right arrows.
- To edit each sub-field, you can use the number keys, or the numpad +/- key. In the case of the time control, you can also increment/decrement it using the up/down buttons which will be appended to right edge of the field.

Update: Build 885.2 - 06 May 2004

INFLD/GUI keyboard handling refinements:

- ALT-DOWN-ARROW may now be used as a keyboard shortcut to display the calendar in the date picker.

- UP/DOWN arrow may now be used, in addition to the numeric keypad +/- keys to increment/decrement the currently selected sub-field (in the date picker).
- SHIFT-TAB now acts like up arrow when TYPE 3 is specified (returning EXITCODE 3).

Update: Build 892 - 10 Jun 2004

(Windows/ATE) Checkboxes and radio buttons can now be created and accessed by **INFLD** similarly to other field types. (In particular, checkboxes correspond well to Y/N fields.) The TYPE code for checkboxes is "|c" and for radio buttons is "|b" (vertical bar followed by a lower case "c" or "b"). If combined with TYPE Y, the field will treat a checked box as equivalent to the single character affirmative (Y, S, O, etc.) for the current language definition, and an empty box as the single character negative (N). Otherwise, a checked box will correspond to a "1" and an unchecked box to a "0".

Note the following idiosyncrasies of checkboxes and radio buttons as **INFLD** field types:

- Checkboxes and radio buttons combine a text field with a square or round box. Although other fields may also associate an edit control with a text prompt, in all other cases, **INFLD** does not concern itself with the text prompt (which much be displayed via a separate operation). When a particular checkbox or radio button is being edited by **INFLD**, the corresponding text will be highlighted via a dotted line, giving the operator a visual indication of what field they are on. (There is no cursor for checkboxes and radio buttons.)
- Because of this association between the text and the box or button, if you want to use **INFLD** to create the control, you must supply the text string (label or prompt) in the SETDEF parameter. (SETDEF does not otherwise apply to these two field types.) (This is not to be taken as a recommendation that you use **INFLD** to CREATE checkboxes and radio buttons; see item D below for recommendation on creating them.)
- In contrast to the case with other field types, in which the field becomes an editable control when it has the focus and is converted back to a static/sunken text control when not being edited, checkboxes and radio buttons remain the same whether they have the focus or not. The only visible difference when they have the focus is the dotted outline described above.
- Because of the above, it is typical for checkbox and radio button fields to be created at the same time that your program normally creates the data entry form background. You can create them for display purposes either using MIAMEX,119, or TAB(-10,20), or **INFLD**. Once created, when you want to edit the field, **INFLD** will locate the existing control, if it exists, by looking at the ROW, COL, and V parameters. (If V is < 0, it is taken as the actual control ID # of a control previously created with XCALL MIAMEX,119, and ROW and COL are ignored. If V > 1, then it is treated as the control ID # of a group box to which the checkbox or radio button belongs, and the ROW and COL parameters are offset within the group box.) Unless V identifies the control directly, the existing controls are scanned to find one which has the same starting row/col, or the same ending row/col. The point of comparing just the starting or just the ending position is that after having created the checkbox/radio button control, you may want to treat it like Y/N field, in which case you would probably be specifying a field of size 1 (even though the full control will be several columns wide due to the text component.)
- When **INFLD** is operating on a checkbox (and the corresponding text is highlighted), you can toggle the check status with the space bar, or by clicking with the mouse. You can use ENTER to exit, or any of the other field exit keys which you've enabled according to the normal **INFLD** rules.

- When **INFLD** is operating on a radio button, the button that has the focus will automatically be selected (as if you had clicked it with the mouse.) You cannot toggle the button with the spacebar (partly because it wouldn't then be clear which other button in the group should be toggled on.) (Because it is a radio button and not a checkbox, only one button in the group, at most will be checked at one time.) Considering this idiosyncrasy of radio buttons, they don't work that well as **INFLD** fields unless your users are prepared to use the mouse to jump to the next non-radio button field. (Otherwise, unless you enable a specific exitcode to jump from the current button to the next group of controls, there would be no way for the operator to keyboard through a group of radio buttons without leaving the last one selected.) Thus, it may make more sense to treat radio buttons more like pushbuttons, i.e., as controls that are not operated on directly by **INFLD** but instead are available for the user to click on at will, with the program querying them before processing the screen data.

When treating checkboxes and radio buttons like normal **INFLD** fields, there may be some confusion over the following similar situations:

- While within an **INFLD** call targeting a specific checkbox or radio button, the field will visibly maintain the focus and all keyboard will be filtered through the control. (The result is that other than space bar and defined exit keys, any other keys will probably be ignored.) Clicking on another field may or may not cause the current field to lose focus, depending on various factors.
- If you click on a checkbox or radio button while another **INFLD** field is active, unless the clicked-on control is set up to transmit an exitcode which will cause the currently active field to exit and the program to jump to the just-clicked field, although the click operation will change the checked status of the checkbox or radio button, the clicked-on control will not retain the focus. Instead, the focus will revert back to whatever field was previously doing input. The point of this is that if you choose not to use **INFLD** to individually edit your checkboxes, you can use a single generic input routine which just waits for you to exit the current screen (presumably by clicking on a button or entering a special code or key). Clicking on individual boxes in this case will change the status of the checkboxes, but not interrupt the current input operation. (You can query the checkboxes or their control variables all at once when you decide to leave or update the screen.)

Update: Build 892.4 - 17 Jun 2004

INFLD now implements fields which have a "normal" SETDEF parameter (containing a list of allowed entries, e.g. ",red,blue,green,purple,,") as combo boxes. User can type or select choice from drop-down box. As before, to allow an empty field, add TYPE O (for optional).

Update: Build 893 - 24 Jun 2004

(Windows/ATE) Checkboxes created with **INFLD** now have the MBF'UNPROTECTED flag set so they can be deleted with TAB(-1,10) (more like static text than like regular buttons).

Update: Build 894 - 28 June 2004

(Windows/ATE) Several improvements have been made to checkboxes (now TYPE ||c), allowing them to be handled nearly like Y/N fields. Contrary to the recommendations set out under edit 892 below, our latest recommendation is to either use **INFLD** for all aspects of checkbox implementation (creation, display,

editing), or use XCALL MIAMEX,119 for all aspects. You can mix the two (i.e. create the checkbox using MIAMEX,119 and edit it with **INFLD**, but this is likely to cause confusion.) If you use the MIAMEX,119 method, then your program does not need to monitor the user while editing the checkboxes; you only need to query the checkboxes at some later time to see which ones are checked. In contrast, if you use the **INFLD** method, then you should implement an exitcode sequence allowing the user to go directly to a checkbox field by clicking on it, and treat it almost just like a normal Y/N field.

The main differences between Y/N fields and checkboxes are:

- The text label of the checkbox is an integral part of the control. You must pass the label in the SETDEF parameter, and set the XMAX value large enough to accommodate the label (instead of just 1 which would be typical for Y/N fields.)
- The user toggles the checkmark with the mouse or the space bar.
- The OFCLR/OBCLR fields in INFCLR are now used to set the color of the checkbox label text when displaying or redisplaying the checkbox.

Note that, at this time, we are only encouraging the use of **INFLD** for checkboxes, and not radio buttons. In most cases, a single combo box (see 892-4 below) is much cleaner to simpler than a series of radio buttons. If you insist on radio buttons, it is probably best to just implement them using the MIAMEX,119 method.

Update: Build 895 - 20 July 2004

(WINDOWS/ATE) GUI version of **INFLD** now obeys the INFCLR settings. Previously, while editing, it always used the standard Windows colors. If you want to keep that behavior, set EFCLR and EBCLR to -2, either in the INFCLR parameter passed to **INFLD**, or in the INI.CLR file.

(WINDOWS/ATE) A new option to "use fixed pitch font in edit boxes" has been added to the Misc Settings dialog. When set, **INFLD** uses the currently defined fixed pitch font (see Settings...Fonts) and attempts to scale it slightly down to fit in the edit window better. It doesn't completely eliminate the complaint that the size of the edit window doesn't perfectly match the number of characters which are allowed, but now the edit windows will more or less uniformly appear to have room for one more character than is allowed. (This is necessary, otherwise Windows won't display that last character without scrolling the field.) Note that when you edit the field it will be redisplayed using the defined "proportional" font. If you don't like the drastic change between the fixed pitch (in edit mode) and the proportional pitch (upon display), then you can always set the "proportional font" to the same as the fixed pitch font, in which case all of the GUI controls will use it.

Update: Build 896 - 20 July 2004

(WINDOWS/ATE) The **INFLD** combo box (activated when you have a "normal" SETDEF list and |G) can now accommodate about 1200 bytes of options in the SETDEF parameter (up from 512 previously). For choosing from lists longer than that, you might as well just use **PCKLST** or **XTREE**.

Update: Build 897 - 10 August 2004

(INFLD GUI) The handling of fixed pitch fonts in the GUI version of INFLD has been improved somewhat. Unlike proportional fonts (which we only scale vertically and let the horizontal size default based on the

display aspect ratio and judgment of the font mapper), fixed pitch fonts can be scaled both vertically and horizontally. We use this to ability to try to generate a font that fits within an INFLD edit box in such a way that the box gets filled up when you get to the maximum number of allowed characters for that field. (With proportional fonts, this is out of the question.) Note that because fixed pitch fonts are currently scaled to an integer width in pixels, as a practical matter, the maximum number of characters is not going to exactly fill the box. But it should be reasonably close. If you can't stand that imperfection, you might have to just give up on the GUI version of INFLD and continue to use the text version.

Note that along with the scaling improvements, we've reduced the degree of automatic expansion of small field lengths. Previously, 1 and 2 character fields were expanded by one character (to allow for font size irregularities.) When the fixed pitch option is set, we now only expand single character fields to 2 column (mainly because the edit window border reduces the available size of the field by so much that a single character field is just too skinny to look good.)

Update: Build 901 - 19 September 2004

(Windows/ATE) When INFLD displays a message box (typically to report an editing format error), the title of the message box is now taken from the 000,001 entry of the SYS:APPMSG.xxx file (where xxx is your language definition, e.g. "USA", "CDN", etc.) If the file or entry is not defined, then it continues to use "INFLD".

INMEMO

```
xcall INMEMO, opcode, text, channel, strow, stcol, endrow, endcol,
     link, xpos, vspec, mmoclr, extctl
```

As with INFLD, A-Shell contains a runtime implementation of MicroSabio's **INMEMO.SBR** (and XFRMMO) which is about 95% compatible with the AMOS version. The intent is to allow applications which were written to use **INMEMO** to continue to run under A-Shell. Licensees of the AMOS version can simply transfer all of the associated **INMEMO** utilities (which were written in AlphaBASIC) from AMOS to A-Shell, and of course the memo files themselves are binary compatible across platforms.

Those without an AMOS license for **INMEMO** will need to contact MicroSabio to obtain the necessary materials for developing with **INMEMO**.

The following condensed documentation may be of use to those who are familiar with the routine but just need a quick reference to the parameter list; refer to the separate INMEMO documentation for more details. Also see MMOHDR.BSI, which defines may of the parameter symbols and values that are used below.

Comments

PCKLST**.SBR** acts as a front-end to INMEMO to create pop-up pick-list boxes.

The A-Shell version of **INMEMO** has some options which can greatly simplify the process of implementing casual notepads in applications. Again, refer to the INMEMO User's Guide

The A-Shell version of **INMEMO** is slightly more advanced that the AMOS version in the implementation of free-form menus, in that it supports hidden text associated with an option (following a backslash as with regular vertical menus). For example, assuming the menu item delimiters were brackets, the text "[Mystery Option\secret]" would display as "Mystery Option", but if selected, would return the full "Mystery Option\secret".

Opcode

opcode (numeric)
specifies a combination of flags from the table below.

mmohdr.def	Description
MMO_DSP	Display
MMO_EDT	Edit
MMO_BDR	Border
MMO_LID	Smart line insert and delete
MMO_DEL	Delete
MMO_LIN	Return one line of memo text
MMO_OPN	Open memo file
MMO_CLS	Close memo file

mmohdr.def	Description
MMO_NBR	Do not redraw border
MMO_NMR	Do not redraw border or text
MMO_SCH	Search
MMO_SIL	Silent operations (no display)
MMO_DPG	Display with paging control
MMO_MNU	Menu mode
MMO_TBL	Table lookup
MMO_FST	Fast menu mode
MMO_EWU	Edit without update
MMO_UWE	Update without edit
MMO_SVA	Save screen area
MMO_RSA	Restore screen area on exit
MMO_CXY	Start at position EXTROW,EXTCOL
MMO_NOA	Do not display navigation arrows
MMO_AAH	Auto adjust (shrink) height to fit data
MMO_IPG	Intelligent paging mode
MMO_DBM	Start display at bottom
MMO_ISL	Insert into sorted list
MMO_APS	Alternate prompt style
MMO_NAF	No auto formatting (wrapping)
MMO_OTX	Output memo to TEXT
MMO_FFM	Free form menu or dialog box mode
MMO_OPT	Disk optimized mode (n/a under A-Shell)
MMO_RET	Return key exits memo
MMO_HDR	Output invisible headers with chr$(26) delimiters

Other Parameters

text (string variable)
has various uses, from titles, to preloaded key sequences, retrieved lines, menu selections, and even entire memo or menu contents.

channel (numeric)
is generally the file channel that the memo file is open on. If the CHANNEL parameter is a string, it will be interpreted as the name of the memo file, in which case the memo file will be automatically opened, closed, and even created, if necessary.

strow, stcol, endrow, endcol (numeric)
specify the dimensions of the memo box, around which the border, if applicable, will be drawn.

link (B,2 or B,4)

is the link to the memo record.

xpos
is a control variable, which should be mapped as follows

```
MAP1 XPOS
   MAP2 POS,B,4
   MAP2 POS2,B,2
```

vspec
controls the maximum scrollable width and height of the memo, and must be mapped as:

```
MAP1 VSPEC
   MAP2 VWIDTH,B,2
   MAP2 VROW,B,2
```

mmoclr
if specified, controls the colors for various parts of the memo display, and must be mapped as follows:

```
MAP1 MMOCLR
    MAP2 BFCLR,B,1        ! border foreground
    MAP2 BBCLR,B,1        ! border background
    MAP2 TFCLR,B,1        ! text foreground
    MAP2 TBCLR,B,1        ! text background
    MAP2 AFCLR,B,1        ! arrows foreground
    MAP2 ABCLR,B,1        ! arrows background
    MAP2 PFCLR,B,1        ! prompt (title) fg
    MAP2 PBCLR,B,1        ! prompt (title) bg
    MAP2 WFCLR,B,1        ! warning mssg fg
    MAP2 WBCLR,B,1        ! warning mssg bg
    MAP2 SFCLR,B,1        ! status line fg
    MAP2 SBCLR,B,1        ! status line bg
    MAP2 RFCLR,B,1        ! ruler/reserved fg
    MAP2 RBCLR,B,1        ! ruler/reserved bg
```

extctl
may optionally be specified for extended control parameters, mapped as follows:

```
MAP1 EXTCTL
    MAP2 EXTCOD,B,2       ! exit code
    MAP2 EXTERR,B,2       ! exit error status
    MAP2 EXTMAP,B,4       ! exit code enable bitmap
    MAP2 EXTROW,B,2       ! cursor row
    MAP2 EXTCOL,B,2       ! cursor column
    MAP2 EXTBYT,B,4       ! # bytes in memo
    MAP2 EXTHIT,B,2       ! # rows
    MAP2 EXTWID,B,2       ! longest row length
    MAP2 EXTPRW,B,2       ! # protected rows
    MAP2 EXTSMI,S,1       ! start menu item char
    MAP2 EXTEMI,S,1       ! end menu item char
    MAP2 EXTMRW,B,2       ! error message row
    MAP2 EXTTOP,B,2       ! top window offset
    MAP2 EXTLFT,B,2       ! left window offset
    MAP2 EXTPCL,B,2       ! # protected columns
    MAP2 EXTTIM,B,2       ! max inter-char timeout
    MAP2 EXTOTH,B,2       ! (A-Shell only) other menu exit keys:
                          ! 02 = left arrow (EXITCODE -40)
                          ! 04 = right arrow (EXITCODE -41)
                          ! 08 = up arrow (EXITCODE -42
```

```
                        ! 16 = down arrow (EXITCODE -43)
                        ! 32 = tab key (EXITCODE -44)
    MAP2 EXTJNK,X,14    ! for expansion
```

Of the above parameters, perhaps the most important are *extcod* and *exterr*, whose values are shown in the tables below. The former is similar to the *exitcode* parameter in INFLD, receiving a code indicating how the memo was exited. The latter indicates various other error conditions. The variable/symbol names in the first column of the tables below are defined in MMOSYM.BSI.

Symbol	Description
MMX_OK	Normal Updated exit
MMX_QUI	Non-Updated exit (Control-C)
MMX_VTS	Non-Updated exit (VSPEC too small)
MMX_TIM	Updated exit (time out)
MMX_TAB	TAB used to select menu item
MMX_SPC	Spacebar used to select menu item
	Command function exits are returned as negative numbers F1 = -1, etc.

Symbol	Description
MME_ERR	abnormal exit (see POS)
MME_SAF	failure to save or restore area
MME_OK	normal exit

ISMROK

```
xcall ISMROK, ch, ida'alc, ida'avl, recsiz, keysiz, keypos, devdrv,
      idx'alc, idx'avl, ida'nxt, idx'nxt
```

ISMROK.SBR reads the "rock" of the traditional ISAM file open under channel *ch* and returns the requested parameters (all of which are optional and can be specified using any numeric data types). It will be left as an exercise to the reader to determine the usage of the above parameters (hint: check out **ISMUTL.LIT**).

ITC

```
xcall ITC, func, type, mesg, retv
```

(Unix/Linux only) **ITC.SBR** allows programs to send and receive messages between jobs.

Parameters

```
MAP1 FUNC,<any numeric type>     ! opcode
MAP1 TYPE,<any numeric type>     ! message type or process ID
MAP1 RETV,B,1                    ! return code (0=ok)
MAP1 MESG                        ! message
   MAP2 MTYPE,B,4               ! internal code
   MAP2 MESGX,X,<any size>      ! message contents
```

func
indicates the operation to perform, as described in the following table.

Value	Meaning
0	Send message to cooperating receiver
1	Receive message (do not wait if not available)
2	Receive message (wait until available)
3	Send message and signal receiver
	Write exactly *count* characters (or up to null if *count*=0)

type
identifies the type of message, or the process ID of the target job to signal. In the case of *func* 3, it must be the process ID of the target job to signal. (In this operation, the contents of the message may not be important; since the signal itself could be used to notify the target job to look for subsequent messages.) For the other operations, it must be a value that both sender and receiver agree on, because the receive operation will ignore messages not sent with the specified *type*. (The idea here is to allow multiple message formats or priorities to be used simultaneously.) The exception is that the receiver may set *type*=0 to receive the first message in the queue, regardless of its type. To avoid confusion between types that are process IDs and types that are message types, it is recommended that you adopt a standard of using message *type* values higher than the range of process IDs (which might extend as high as 2**24 on very large systems. *Type* should also be below 2**31 to avoid confusing them with negative *type* numbers, which are used to request to read any message whose *mtype* is less than or equal to the absolute value of the specified *type*.

mesg
contains the message to send or receive. Obviously, the communicating programs need to agree in advance on the layout corresponding to each message type. It must begin, as shown above, with a 4 byte field which is reserved for internal use by the subroutine. (Actually, it plugs the *type* value into *mtype* for outgoing messages, and ignores any incoming messages whose *mtype* does not match the *type* specified.)

retv
returns a 0 or 1 indicating the success of the operation (0 = success).

Comments

UNIX/Linux systems do not allow one user to signal another unless they are both share the same effective user ID or if the sender is root. For this reason, some sites set the "setuid" flag on the ashell executable to make all A-Shell users effectively the same user, or else they rely on the system administrator being able to switch to root in order to use operations which require signaling other users. Such operations include ITC.SBR FUNC=3, KILL.LIT, CHAT.LIT, SEND.LIT, FORCE.LIT, and JSTAT.LIT.

JOBCMD

```
xcall JOBCMD, command
```

A-Shell emulates the AMOS feature of being able to specify a command string which is automatically executed whenever the job returns to the AMOS command (or dot) prompt. Setting the command to a null string disables the feature, which effectively makes it possible again for the user to interactively type commands when at the AMOS prompt. This feature is often used to force users back into some kind of a menu in case they somehow manage to drop out of a program. You have to be careful with this facility, since once you set it, there is no way to exit except by making another call to **JOBCMD.SBR** with a null string as the argument.

The maximum length of the Command string is 29 characters plus a trailing null byte.

JOBDAT

```
xcall JOBDAT, job'info
```

JOBDAT.SBR is yet another example of a subroutine which returns a variety of information about the current job. This one seems to have been connected with the inSight juggernaut.

Parameters

```
MAP1 JOB'INFO
     MAP2 JD'SYSNAM  ,S,16              ! OS name, e.g. "Windows/32"
     MAP2 JD'JOBNAM  ,S,6               ! Job name
     MAP2 JD'JOBNUM  ,F,6               ! Job number
     MAP2 JD'JOBLOG  ,S,20              ! e.g. "DSK0:[150,277]"
     MAP2 JD'USRNAM  ,S,20              ! User login name
     MAP2 JD'PRGNAM  ,S,6               ! Program name
     MAP2 JD'PRGVER  ,S,16              ! Program version
     MAP2 JD'TRMROW  ,F,6               ! Current # display rows
     MAP2 JD'TRMCOL  ,F,6               ! Current # display cols
     MAP2 JD'TRMFGC  ,F,6               ! Foreground color #
     MAP2 JD'TRMBGC  ,F,6               ! Background color #
     MAP2 JD'TRMFLG(32),B,1             ! Terminal flags (see below)
```

Each entry in the JD'TRMFLG() array corresponds to a terminal feature from the table below. The array item is set to 1 if the feature is supported by the current display device, else 0.

Item	Meaning		Item	Meaning
1	Alternate page		16	Line insert
2	Block fill		17	Auxiliary printer port
3	Column insert		18	80/132 column
4	Color		19	Smooth scroll
5	Multi-byte* translations		20	Box commands
6	Alpha terminal		21	Mode attributes
7	Addressable status lines		22	AM70-style color commands
8	Split screen		23	No-space attributes
9	Erase to end of line		24	"True" graphics
10	Erase to end of screen		25	Gray scale
11	Underscore		26	Proportional space fonts
12	Blink		27	Variable rows
13	Dim		28	8 bit support
14	Reverse video		29	Insight Support
15	Character insert			

* (function key)

JOBNUM

```
xcall JOBNUM, jobnumber
```

JOBNUM.SBR returns the current job number (in any numeric parameter format).

Note that the job number is established when the session is launched, as the position within the job table, and remains fixed for the duration of a session.

JOBTRM

```
xcall JOBTRM, jobnam, trmnam, tdvnam {,ctljob, ctltrm, ctltdv}
```

JOBTRM.SBR returns the job name, terminal name, and terminal driver name of the current job. (All parameters should be mapped as strings of 6 or more characters.) Optionally, it can also return the same set of information for the "controlling job". The idea of the "controlling job" is mainly applicable in AMOS multitasking environments (such as MULTI and PolyTrack) where the program wants to know the identify of the "controlling" (or "real") job in order to associate some resources with the user or terminal. In the A-Shell environment, the "controlling job" will, for the purposes of this subroutine, always be the same as the current job, except in the case of PolyShell, in which case the controlling job name will be the same as the current job, minus the last character. (For example, the PolyShell control job PSHAA will spawn user jobs PSHAA1, PSHAA2, etc.)

The terminal name under A-Shell is virtually always the same as the job name.

JULCVT

```
xcall JULCVT, sepdat, juldat, flag
```

JULCVT.SBR converts between a separated date format and Julian date format.

Parameters

sepdate (B,4)
is the date to convert to Julian (if *flag* = 0), or the return date (if *flag* = 1). It is in AMOS "separated" format, which is as follows:

```
MAP1 Sepdat,B,4
MAP1 SepdatX,@Sepdat
    MAP2 Mon,B,1                        ! Month (1-12)
    MAP2 Day,B,1                        ! Day of month (1-31)
    MAP2 Yr,B,1                         ! Offset from 1900 (2003=103)
    MAP2 Dow,B,1                        ! Day of week (Mon=0, Sun=6)
```

juldat (numeric)
is the Julian date equivalent of *sepdate* (if *flag* = 0) or the date to convert to separated format (if *flag* = 1). This is the number of days since "The Beginning" (about 4716 BC).

flag (numeric)
should be set to 0 to convert from *sepdate* to *juldat*, else 1 to convert the other way.

Comments

The AlphaBASIC system variable `DATE` can be used to set *sepdat* to the current date (i.e. *sepdat* `= DATE`).

Also see **GRECNV** and **DATES** for other date conversion routines.

LOG

```
xcall LOG, logstr {,qflag {,status}}

xcall LOG, dev$, p$, pn$
```

LOG.SBR allows you to log to a new disk and PPN, using an ersatz or traditional specification, with an option to silence the normal output (start message) that you would otherwise see with an AMOS LOG command. It may also be used to return your current device, project, and programmer number.

Examples

```
xcall LOG,"SYS:"              (log to SYS:)

xcall LOG,"LOG DSK0:[1,4])    ('LOG' is optional)

xcall LOG,"SYS:","Q"          (silence output)

xcall LOG,DEV$,P$,PN$         (return current login)
```

Comments

If the *status* parameter (floating point type) is specified and is non-zero, the routine attempts to check if the operation succeeded and returns a zero for success or the system errno for an error. Whether or not *status* is specified, LOG always checks to see if the destination device is defined. However, without *status*, it does not check if a PPN within a device actually exists and is accessible. This it could be possible to appear to log to non-existent ppns.

LSTLIN

xcall LSTLIN, cmdline

LSTLIN.SBR may be used within a BASIC program to retrieve the command line from which the program was launched. This is useful for allowing command line parameters to be passed to BASIC programs, and nearly essential for allowing BASIC programs to emulate AMOS LIT commands. It is similar to the BASIC Plus CMDLIN (or CMDLIN$) system variable, except for the following: 1) **LSTLIN** does not fold the command line to upper case, like CMDLIN does; and 2) **LSTLIN** includes the name of the current program, whereas **CMDLIN** only includes the part of the command line after the program name. The following example will make this distinction clear:

```
Program LSTLIN,1.0(100)
MAP1 A$,S,100

? "CMDLIN$ = [";CMDLIN$;"] (BASIC Plus only)"
? "CMDLIN = [";CMDLIN;"] (same as CMDLIN$)"
xcall LSTLIN,A$
? "XCALL LSTLIN = [";A$;"]"
```

If the program above is compiled with COMPLP (or COMPIL /X:1), the command line below will produce the following output:

```
.run lstlin hello world
CMDLIN$ = [HELLO WORLD] (BASIC Plus only)
CMDLIN = [HELLO WORLD] (same as CMDLIN$)
xcall LSTLIN = [lstlin hello world]
```

Note that **LSTLIN.SBR** works in all versions of BASIC, while use of the CMDLIN or CMDLIN$ system variable requires BASIC Plus (complp or compil /x:1).

MATCH

```
xcall MATCH, rtncod, value, range
```

MATCH.SBR is useful in building flexible record selection logic. With it, you can easily allow the users to specify a list of acceptable values, or ranges of values, for a particular field. Then for each candidate record, just pass the value and range specification to the subroutine and let it worry about numeric, string, and date comparisons.

Parameters

rtncod (numeric)
returns the results:

Value	Meaning
0	No match; value is not within the specified list or range
1	Match; value is within range
-1	Parameter error
-2	Value type not supported
-3	Range string too long
-4	Syntax error in range

value (type F, S or B)
is the value which you want to check against the range.

range
is the range specification in the form:

```
r1,r2,r3,...,rn
```

Each of the *rx* entries in the range string may be either a single value or a from-to range expressed as a beginning and ending value, separated by a dash. Each item in the range list is interpreted according to the type of the *value* parameter. String entries are case sensitive. If both *value* and *range* are of type string, and *value* is in the form:

```
mm/dd/ccyy
```

then each entry in the *range* string is interpreted as a date in a similar format. `mm/yy` is interpreted as `mm/01/ccyy` if followed by a dash, or `mm/31/ccyy` if preceded by a dash. The `cc` value is assumed to be 19 unless specified or unless the SBR=CCYY:## parameter in the MIAME.INI has been established.

On return, *range* is converted into a "compiled" format to eliminate the need for parsing it out on the next call. (If used for file record selection, you would typically be calling this routine hundreds or even thousands of times, with different *value* parameters but the same (or a limited number of) *range* strings.

A NULL *range* string matches all, as does a NULL *value* (for String ranges). Examples:

Range String	Meaning
"AA-CZ,F,JA*"	Match any string starting with “AA” thru “CZ” (“ABCD” matches, “CZ1” does not). Also match if Value = “F”, or if it equals anything starting with “JA”.
"91300-91399,93132,95555"	Match any number 91300 thru 91399, or match 93132 or 95555.
"03/15/97,04/97-2/98,3/02/1999"	Date match, including Mar 31 1997, or April 1 1997 thru Feb 28 1998, or March 2, 1999.

MIAMEX

```
xcall MIAMEX, opcode {,param1 {,param2 .. {,paramn}}}
```

MIAMEX is a collection of over 100 utility functions that are gathered together here for efficiency or because they do not fall nicely into other categories. Each of the functions is listed below, with detailed documentation on the following pages. Interested programmers should also consult the sample program, MIAMEX.BAS, which can be found in the DSK0:[7,376] directory and which contains examples of the more generally useful opcodes. In all of the MIAMEX specifications, the first parameter is *opcode,* as specified using the variable name (MX_xxx) defined in **Error! Reference source not found.**. These opcodes are shown in the following table.

Value	Function
1	Set / return command file status
2	Exit A-Shell
3	Translate AMOS filespec to native
4	Get Control-C status
5	Get echo status
6	Set command prompt
7	Get hex output status
8	Set / reset hex mode
9	Octal conversion (OCVT)
10	Check for ppn
11	Return ersatz
12	Return A-Shell version
13	Set / return WSET flags
14	VUE
15	COMPIL
16	CPRE
17	Synchronize A-Shell PPN with system path
18	Get next command file line
19	Get next device
20	Find first matching file
21	Find next matching file
22	Get directory separator
23	End directory processing
24	Get prompt
25	Get hash
26	Print filespec
27	Copy file

Value	Function
28	Output error message
29	Output rename error message
30	Return queue block
31	Return environment variable value
32	Set default extension
33	Match cmdlin
34	Get CISAM/DISAM version
35	Set crm
36	Get hex status
37	Set / reset AMOS hash flag
38	Create directory
39	Delete directory
40	Is ODBC available?
41	Zap queue
42	Return first ODBC table
43	Return next ODBC table
44	ODBC end
50	Return LOKSER flags
51	Set LOKSER flags
52	Return CMDINP flags
53	Set CMDINP flags
54	Return signal received flags (UNIX only)
55	Clear signal received flags (UNIX only)
56	Change TERM environment variable (UNIX only)
57	Kill job
58	Display license
59	Return OPTIONS flags
60	Set OPTIONS flags
61	Return TRACE flags
62	Set TRACE flags
63	Set function key translation or PFK
64	Return PFK name
65	Take screen snapshot
66	Force queue rebuild
67	Lock queue
68	Unlock queue
69	Return umask (UNIX only)
70	Set umask (UNIX only)

Value	Function
71	Add menu item (Windows only) (same as AUI, “MENU”)
72	Display LITMSG.xxx message
73	Return jobtbl.sys record
74	Write jobtbl.sys record
75	Get current login time
76	Launch Telnet server mode (Windows only)
77	Display/hide Window (Windows only)
78	Get global DO parameters
79	Set global DO parameters
80	Reserved
81	Get / Put **Error! Reference source not found.** parameters
82	Exit to dot prompt (even from with an **Error! Reference source not found.**)
83	Get FKEYWAIT value (Unix only)
84	Set FKEYWAIT value (Unix only)
85	Get / Set Window title
86	Get operating system error message by errno
87	Disable stream file buffering
88	Flush stream file buffer
89	Clone file channel (within**Error! Reference source not found.**) (similar to M68 FLSET)
90	Get / Set bevel options (Windows only)
91	Define BG color # to track system COLOR_3DFACE (Windows only)
92	Enable / Disable PolyShell hot keys
93	Get license info (used by ABOUT.LIT)
94	Get current cursor position
95	Windows Open File dialog (Windows only)
96	Shell Execute (open/print registered file type) (Windows only)
97	CreateDirectory (Windows only)
98	Start ATE
99	Retrieve registry value (Windows only)
100	Play a sound file (Windows only)
101	Get swap wait value (in ms) (Unix only)
102	Set swap wait value (in ms) (Unix only)
103	Get / set ashell command line switches
104	Get process ID (pid)
105	Get / set clipboard (Windows only)
106	Get operating system version
107	Get USRMEM module information

Value	Function
108	Load module (or variable) into USRMEM
109	Delete module from USRMEM
110	Save module from USRMEM to disk
111	Read / write USRMEM data directly
112	Set AutoMouse translations (Windows only)
113	Change user partition size
114	Marshall parameters for remote XCALL
115	Indirect XCALL
116	Process an INI.CLR file
117	MAPI Send Mail (Windows only)
118	Get / set file pointer within stream (input or output sequential file)
119	Manage Windows buttons (Windows only); same as AUI, “Control”
120	Prompt for Windows printer (Windows only)
121	Get / set chain-to on privilege violation
124	Ouput STRING message to ashlog.log
125	Retrieve information about last mouse click
126	Sink / unsink specified box
127	Get / set rounding factor
128	Get IP address
129	Get / set GUI-related flags; same as AUI, “Environment”
130	Retrieve startup command
131	Retrieve stats for specified path
132	Reformat filetime from MIAMEX'131
133	Expand a file in place
135	Eventwait
137	Windows help system
138	Registry operations
141	Set parent controls
142	Retrieve CMD file variables
143	Get/set INFLD global TYPE codes
144	Get/set DEBUG level
145	<reserved>
146	Retrieve last line number of program
147	Count lines, max line length in file

MIAMEX 1: Set/get command file status

```
xcall MIAMEX, MX_CMD_STATUS, cmdcode
```

This command gets or sets command file status. *cmdcode* is a floating point variable which contains the new command file status to set, and into which the current status is returned. Each possible command file status (**:R**, etc.) is given a different numeric code from the table below:

Value	Meaning
0	No command file currently running
1	Command file is in silent (:S) mode
2	Command file is in response (:R) mode
3	Command file is in trace (:T) mode
4	Command file is executing last line while in silent mode

MIAMEX 2: Exit A-Shell

```
xcall MIAMEX, MX_EXIT
```

This function causes an immediate exit from A-Shell. Control is returned either to the host machine command level, or to a batch or shell-script file if one is running.

MIAMEX 3: Perform FSPEC on AMOS string

```
xcall MIAMEX, MX_FSPEC, spec, local, ext, code, ddb, status

MAP1 SPEC,S,40       ! AMOS file specification
MAP1 LOCAL,S,160     ! Local (native OS) equivalent fspec
```

This function is used as an approximation to the FSPEC monitor call. It takes an AMOS format file specification or device name (*spec*), and processes it into its constituent parts of Device name, File name, and extension *(ddb*). It also optionally returns the host machine pathname corresponding to the AMOS file specification in *local*.

Parameters

spec

should contain the AMOS-format file specification that you want to parse or convert to native format. If the device and/or PPN are omitted, they will be supplied by your current login directory. If the extension is omitted, it will be supplied by the *ext* parameter. Note that the file does not need to exist. (This is useful when you are only interested in determining the native directory equivalent to your current login PPN.)

local

is only used if you wish to return the host machine pathname for the file specification given in *spec*. This is specified by the `FS'TOH` bit in the code parameter, *code*. If it is to be used, then the variable

must be defined to be a string of at least 100 characters in length. In all other cases, the parameter may be specified as a null string ("").

ext
is the default extension to use when processing a full file specification. It will be used if the given specification contains only a filename and no extension. The extension used is returned in the extension field of the *ddb* structure variable, and is used in the host format pathname if one is returned.

code
is used to determine exactly how the file specification processing is to take place. It is bit-mapped, and each bit may be set by adding together the symbolic values defined in the include file, **Error! Reference source not found.**. The `FS'FMA` (2) bit indicates that the file specification given in the *spec* parameter is to be used. Otherwise the already processed specification given in its constituent parts in the *ddb* structure parameter will be used. It only makes sense to omit this bit if the `FS'TOH` (4) bit is set, indicating that a host machine pathname is to be returned.

fs'toh
indicates that the host machine path name corresponding either to the file specification in *spec* or in the *ddb* structure parameter is to be returned in the *local* parameter. If this bit is not specified then *local* may be given as a null string ("").

fs'drv
indicates that the drive name and number only are to be processed. When used in conjunction with *fs'fma* (2), the device specification given in *spec* is returned in the *ddb* structure parameter.

ddb
is a structure defined in the include file, **Error! Reference source not found.**, and is defined as follows:

```
MAP1 DDB
    MAP2 DEV,s,6          ! 6 characters AMOS device
    MAP2 FILNAM,s,32      ! 6 characters AMOS or 32 host
    MAP2 EXT,s,8          ! 3 characters AMOS or 8 host
    MAP2 PRJ,s,3          ! 3 character justified P
    MAP2 PRG,s,3          ! 3 character justified PN
```

It is used to receive either the output of the function call if *fs'fma* is specified, or the input to it if *fs'toh* is specified, or both. Note the size of the *filnam* and *ext* fields. If the *fs'fma* bit is specified, then the AMOS format file specification passed, can actually be in the format of a local pathname (with long filename), in which case the filename and extension will be processed correctly.

status
is a floating point variable into which the return code of the function call is placed. It will only be non-zero if the device name in *spec* or the *ddb* structure does not exist and *the fs'toh code bit is specified*.

MIAMEX 4: Get Control-C status

xcall MIAMEX, MX_GETCTRLC, ccon

The *ccon* parameter is a floating point field into which a non-zero value is returned if Control-C interrupts are currently enabled, and a zero value otherwise.

You can enable/disable Control-C interrupts using XCALL CCON, or SET.LIT.

MIAMEX 5: Get echo status

```
xcall MIAMEX, MX_GETECHO, echon
```

The *echon* parameter is a floating point field into which a non-zero value is returned if terminal echo is currently enabled, and a zero value otherwise.

You can enable/disable echo using XCALL ECHO and XCALL NOECHO.

MIAMEX 6: Set command prompt

```
xcall MIAMEX, MX_SETPROMPT, prompt
```

The *prompt* parameter is a string field of up to 20 characters which should be set to the desired comment level prompt string (i.e. to be used in place of the default "dot" prompt).

You can also set the command prompt with SET.LIT. See MIAMEX 24: Get command prompt to get the current prompt.

MIAMEX 7: Get hex output status

```
xcall MIAMEX, MX_GETHEX, hexon
```

The *hexon* parameter is a floating point variable which will be set to a non-zero value if hex mode is currently set. Otherwise (i.e. if octal mode is set) it will be set to zero.

MIAMEX 8: Set/reset hex mode

```
xcall MIAMEX, MX_SETHEX, hexon
```

This function sets numeric output to hex or octal. The *hexon* parameter is a floating point variable which should be set to a non-zero value to enable hex mode; else it will enable octal mode.

You can also set hex or octal output mode with SET.LIT. The most commonly used utility that is affected by this is DUMP.LIT.

MIAMEX 9: Output hex or octal number

```
xcall MIAMEX, MX_OCVT, number, size, flags {,string}
```

Symbol	Description
OT_ZER	Disable leading zero blanking
OT_TRM	Output to terminal
OT_MEM	Output to memory
OT_LSP	Output leading space
OT_TSP	Output trailing space
OT_OCT	Force output in octal

This operation is the functional equivalent of the AMOS **OCVT** monitor call, and is used to output the number in *number* formatted in either octal or hexadecimal. The base used for output is that set and monitored with MIAMEX functions eight and nine. (Or achieved with the A-Shell **SET HEX** or **SET OCTAL** commands.) The *size* parameter determines the number of digits in the output result. A zero specifies a floating format in which the number of digits used is just enough to fully contain the result. A non-zero size specifies a fixed number of digits for the result with leading zeros being replaced by blanks. In either case at least one non-blank character will always be output.

The *flags* parameter is used to determine exactly how the numeric value is to be formatted. It is bitmapped, and each bit may be set by adding together the symbolic values defined in the include file, **Error! Reference source not found.**.

The *ot_zer* bit disables leading zero blanking. In other words, when a non-zero number is to be printed, leading blanks are converted into spaces. The *ot_trm* bit should always be specified, and is included for compatibility reasons. The *ot_lsp* and *ot_tsp* bits output an additional leading space and trailing space respectively.

The *ot_mem* bit indicates that the output is to be stored in memory (and not output directly to the terminal). If this bit is specified, then the fifth parameter (*string*) must be present, which is a string or unformatted variable into which the formatted numerical output is returned.

MIAMEX 10: Check for PPN

```
xcall MIAMEX, MX_CHKPPN, dev, proj, prog, valid
```

This operation checks for the existence of the specified PPN on the specified device. The *dev* parameter is a string containing the name of the true device and drive (not an ERSATZ device) to be checked. The *proj* and *prog* parameters are numeric values which form the PPN to be checked [*proj,prog*]. Note that, due to AMOS convention, PPN values are in octal, so if for example *proj* contains 10, and *prog* contains 20 (decimal), the PPN looked for will in fact be [12,24].

The *valid* parameter is a floating point variable into which a non-zero value is returned if the PPN exists on the specified device, and zero otherwise.

This operation checks for the host machine pathname corresponding to the requested device and PPN, as set up in the Environment definition file, MIAME.INI.

MIAMEX 11: Get ersatz information

```
xcall MIAMEX, MX_GETERSATZ, ersatz, dev, proj, prog, status
```

This operation is used to obtain information about the defined ERSATZ devices. The *status* parameter is a 6-byte floating point in which a value of one is returned if the end of the ERSATZ list is reached, in which case the rest of the parameters do not contain valid data. On the next call, information on the *first* ERSATZ device will be returned, and then the second, and so on. In these cases the value of *status* will be zero.

Note that a short-cut is available to reset to the first ERSATZ device. If the *ersatz* parameter is a null string, then information on the first ERSATZ device will be returned; it is not necessary to go around a loop waiting for a returned status of one.

Parameters

ersatz should be a six-character minimum string into which will be placed the ERSATZ device name.

dev should be a six character string to contain the real device.

proj and *prog*, should be two- byte binary fields to contain the project and programmer number respectively. See the decription of these parameters for MIAMEX function 19 below for notes on decimal vs. octal and backwards compatibility with the old octal PPN system which was upgraded to decimal in Build 897.

MIAMEX 12: Get A-Shell version

```
xcall MIAMEX, MX_GETVER, version
```

This function returns the current version of A-Shell. The *version* parameter should be a string or unformatted variable large enough to hold the return string (i.e. mapped at least 25 characters), which is in the format: **A-Shell version #.#(###)**.

Use GETVER to return the version of the currently running BASIC program.

MIAMEX 13: Get/set Tracker flags

```
xcall MIAMEX, MX_WSET_status, switches {,op {,ztver}}
```

This operation is used by the A-Shell **WSET.LIT** command program in order to control operation of the Tracker. It provides functionality not available under AMOS—namely the ability to modify the Tracker emulation's under program control.

Parameters

switches

(combine) is a bit-mapped floating-point parameter whose values are as follows:

Value	Description

Value	Description
1	132 column mode disabled
2	Color disabled
4	Field attributes on mode devices disabled
8	Force field emulation (Windows only)
16	ZTERM has been detected
32	ZTERM was probed for but not detected
64	Screen area save/restore disabled

op (numeric)
may be specified as 0 to just retrieve the current settings, or 1 to update the current settings and retrieve the original settings (back into the *switches* parameter). If not specified, 1 is assumed.

ztver (S,10+)
will receive the ZTERM build # if applicable. Note that the format will be something like "ZV2.0.143a." (The ZV prefix is constant for builds 74+.)

MIAMEX 14: Call into VUE editor

```
xcall MIAMEX, MX_VUE, filename, gostring
```

This is used by the A-Shell **VUE.LIT** program to invoke the A-Shell VUE editor (which is coded in C), but could be used by any program requiring full text editing facilities.

filename is the name of the file (file specification or host pathname) to be edited. You may append any of the valid VUE.LIT switches (/R, /Y, /C, /W) to the end of the FILENAME parameter, but you must convert them to "UNIX format", i.e. use a dash instead of a slash and make them lower case. For example:

```
FILENAME = "MYFILE.TXT /W /R"     ! (wrong)

FILENAME = "MYFILE.TXT -w -r"     ! (correct)
```

If the **GO** command is used to exit the editor, then *gostring* (a string parameter of reasonable length) contains a list of commands to be executed, otherwise it is a null string. It would be up to the calling program to process the *gostring*, presumably by chaining to it as an in-memory command file.

If you only want to display a file but not allow editing of it, then you should specify the –r (read only) switch, or better yet, use XCALL EZTYP (which see).

MIAMEX 15: Call into BASIC compiler

```
xcall MIAMEX, MX_COMPIL, filename, switches
```

MIAMEX function 15 is used to implement the A-Shell command programs **COMPIL.LIT**, **OCMPIL.LIT**, **PRE.LIT** and **OPRE.LIT**, by calling into the A-Shell BASIC compiler which is coded in C.

filename is the AMOS file specification or host pathname of the program to be compiled – the compiler itself will default the extension to BAS if necessary.

switches (combine) is a bitmapped floating point parameter specifying the COMPIL command-line functionality:

Value	Corresponding COMPIL.LIT switch or variation
1	/A (use 24-bit instead of 16-bit transfer addresses)
2	/M (treat unmapped variables as errors)
4	/O (omit line numbers from compiled program)
8	Act as OCMPIL (instead of COMPIL)
16	/S (silent output – omit ++include lines)
32	/N (omit phase two compilation statistics)
64	/X:1 (act as COMPLP – BASIC Plus mode)
128	/D (limited d/BASIC support)
256	/X:2 (A-Shell extensions)
512	/V:1 (BASIC 1.4a compatibility)
1024	/T (Trace – output source while compiling)
2048	/I (assume old ISAM in ambiguous OPEN statements)

See the documentation for COMPIL.LIT for further information about the switches and variations of COMPIL.

MIAMEX 16: Call into PREBAS pre-compiler

```
xcall MIAMEX, MX_PREBAS, filename, switches
```

This is equivalent to MIAMEX function 15 except it calls the PREBAS compiler instead.

MIAMEX 17: Synchronize directories

```
xcall MIAMEX, MX_SYNC_CWD
```

Although A-Shell directories generally correspond one-to-one with native directories on the host operating system, it is not automatically the case that the operating system's idea of your current working directory will always match up to your A-Shell login directory. MIAMEX function 17 may be used at any point to resolve this problem by synchronizing the current working directory with that of the current PPN. (This is generally only of interest when interfacing to some host operating facility or executing a native operating system command.)

A call to this function is automatically made by LOG.LIT and by HOST.LIT in order to minimize problems. If a different utility is written to change PPN or execute a host command, it might be advantageous to synchronize the working directory.

See MIAMEX 31: Get environment variable for a way to query the current host operating system working directory.

MIAMEX 18: Get next line of command file

```
xcall MIAMEX, MX_NXTCMD, cmdline, status
```

This operation returns (and in doing so, by-passes) the next line of the current command (**.CMD** or **.DO**) file. It is used in the implementation of **GOTO.LIT** in order to scan the command file for labels.

cmdline should be a string variable long enough to return any expected command line. *status* is a floating-point return code indicating the success of the operation. A zero value indicates that either no command file is running, or the end of the current command file has been reached. A non-zero value indicates success, in which case *cmdline* contains the text of the command line.

MIAMEX 19: Get device definitions

```
xcall MIAMEX, MX_GETDEV, dev, proj, prog, drbase, status
```

This operation is very similar to MIAMEX 11, but is used to retrieve information about the real defined devices, and not the ERSATZ devices. These are the devices defined with the **DEVICE=** command in the MIAME configuration file, MIAME.INI.

The *status* parameter is a 6-byte floating point in which a value of one is returned if the end of the device list is reached, in which case the rest of the parameters do not contain valid data. On the next call, information on the *first* device will be returned, and then the second, and so on. In these cases the value of *status* will be zero. As with the **MX'GETERSATZ** operation, a short-cut is available to reset to the first device. In this case, if the *dev* parameter is a null string, then information on the first device will be returned; it is not necessary to go around a loop waiting for a returned status of one.

dev is a six-character string into which the device name (comprising drive name and drive number) is returned.

As of Build 897, *proj* and *prog* may be either single or double byte binaries. If the device definition is one for an explicit PPN, then these will contain the project number and programmer numbers respectively. If single byte binaries are specified, the PPN information will be encoded in octal (with a range of 0-377). (This is for backwards compatibility with the old octal PPN system.) Any new programs should use two-byte binaries for the *proj* and *proj* parameters, in which case they will be returned encoded as decimal, ready for printing or converting to strings without the need for any octal-to-decimal logic.

drbase should be a 100+ character string into which the base pathname for the device or device/PPN combination is returned.

This pathname will terminate in a directory separator character.

For more information, see the *A-Shell Setup Guide* for the DEVICE parameter of the MIAME.INI file.

MIAMEX 20: Get first matching file

```
xcall MIAMEX, MX_FINDFIRST, path, status, filename, size, attrib
      {,cdate, ctime, udate, utime {, adate, atime}}
```

This operation, combined with its sister function, **MX_FINDNEXT**, is used to scan directories for subdirectories or files.

Parameters

path, string

On entry, *path* must be set to the path which is to be scanned for files. This path must include the terminating directory separator. On exiting, the floating-point status variable, *status*, contains zero to indicate success (and the remaining parameters are set accordingly for the first file found). A non-zero *status* indicates an error, most likely that no files or directories match the specification you gave in *path*.

filename

should be a string which receives the name and extension of the file. It will truncated to fit, if necessary; recommended size is at least 42 characters.

size

should be a floating point field into which the size of the file (in bytes) is written.

attrib

is a bit-mapped floating point field into which are placed the various attributes of the file or sub-directory represented by *filename*, according to the table below:

Value	Meaning
1	Normal file
2	Subdirectory
4	You (current user) have read permission for this file or directory

cdate, *udate*, and *adate* (S,10+)

are the create/status change date, update/modify date, and access date of the file or directory, using the format dd-mon-yr (e.g. “01-Jan-03”). Note that under UNIX systems, the CDATE will be changed when the privileges or ownership of the file is changed.

ctime, *utime*, and *atime* (S,5+)

are the create/status change time, update/modify time, and last access time for the file or directory, using the format (hh:mm).

Comments

Under Windows, you can actually set FILENAME to a literal or wildcard filename to skip directly to the first matching file. However, this is not recommended, since under UNIX you must specify a directory (not a file path) with a trailing directory separator.

MIAMEX 21: Find next matching file

```
xcall MIAMEX, MX_FINDNEXT, status, filename, size, attrib {,cdate,
      ctime, udate, utime {, adate, atime}}
```

This function must be preceded by a call to MX_FIND_FIRST. It returns the next file or subdirectory in the path specified in the original call to MX_FIND_FIRST. On exiting, the floating-point status variable, *pstatus,* contains zero, in which case the remaining parameters are set as for the return from MX_FIND_FIRST, or non-zero to indicate an error (most likely no more files or sub-directories in the path).

Note that the parameter list for this call is identical to that for the MX_FIND_FIRST call, except that the PATH parameter is omitted. The starting path is set by the MX_FIND_FIRST call and maintained internally after that.

See MIAMEX 23: End directory search for important information related to directory scanning.

MIAMEX 22: Get directory separator char

```
xcall MIAMEX, MX_DIRSEP, character
```

This function returns a single byte parameter (usually a string) into which the directory separator character is placed. This is the character used in specifying host pathnames, and is usually a forward slash (/) for UNIX systems, and a backslash (\ ) for Windows systems. Knowing this is useful when parsing native filespecs and also as a quick way of determining which of the two classes of operating systems you are running on.

MIAMEX 23: End directory search

```
xcall MIAMEX, MX_FINDEND
```

The A-Shell directory scanning system only permits the scanning of one directory at a time with the use of the MX_FINDFIRST and MX_FINDNEXT functions. Once the processing of that directory has been completed, then the MX_FINDEND function must be used to terminate and free up any data areas associated with the scan. The recommended logic for scanning a directory is:

```
xcall MIAMEX, MX_FINDFIRST, path,status, filename, size,attrib

LOOP:
if ATTRIB <> 0 goto FINISHED
....
.. Process directory item here
....
xcall MIAMEX,MX_FINDNEXT,STATUS,FILENAME,SIZE,ATTRIB
goto LOOP
FINISHED:
xcall MIAMEX,MX_FINDEND
```

MIAMEX 24: Get command prompt

```
xcall MIAMEX, MX_GETPROMPT, prompt
```

This function returns the current A-Shell command prompt string.

prompt should be a string variable large enough to hold the longest possible prompt (20 characters).

See MIAMEX 6: Set command prompt to set the current prompt.

MIAMEX 25: Get file hash total

xcall MIAMEX, MX_HASHFILE, path, hash1, hash2, hash3, hash4

This function returns the hash total for the specified field.

path must be set to the native (host operating system) format file specification of the file to get the hash off. (See MIAMEX 3: Perform FSPEC on AMOS string to convert an AMOS fspec to a native fspec.) *hash1* thru *hash4* should each be a floating point, to receive the four parts of the hash. The following example illustrates the use of this function to display a hash total in the format used by DIR/H:

```
MAP1 HSHSTR,s,4 ! Octal string byte
MAP1 STRING'HASH,s,15 ! Display format hash-total (###-###-###-###)

xcall MIAMEX,MX_HASHFILE,THISIPATH,HASH1,HASH2,HASH3,HASH4
xcall MIAMEX,MX_OCVT,HASH1,3,OT_MEM+OT_OCT,HSHSTR
STRING'HASH = HSHSTR+"-"
xcall MIAMEX,MX_OCVT,HASH2,3,OT_MEM+OT_OCT,HSHSTR
STRING'HASH = STRING'HASH+HSHSTR+"-"
xcall MIAMEX,MX_OCVT,HASH3,3,OT_MEM+OT_OCT,HSHSTR
STRING'HASH = STRING_HASH+HSHSTR+"-"
xcall MIAMEX,MX_OCVT,HASH4,3,OT_MEM+OT_OCT,HSHSTR
STRING'HASH = STRING'HASH+HSHSTR
print STRING'HASH
```

Note that for display purposes, the standard AMOS and A-Shell hash-total formats treat the numeric values as octal.

MIAMEX 26: Format last file spec

xcall MIAMEX, MX_PRFSPEC, string

This function retrieves the last processed file specification and formats it in standard AMOS format (i.e. dev#:file.ext[p,pn] where dev# and [p,pn] are omitted if the file is on the current device or in the current directory.

As a practical matter, the last processed filespec will be the one last processed via MIAMEX 3: Perform FSPEC on AMOS string.

MIAMEX 27: Copy file

xcall MIAMEX, MX_COPYFILE, ch'in, ch'out

This function copies the file which is open on *ch'in* to *ch'out*. It is left to the calling program to open these file channels, either as sequential files or as random files, and to close them after the operation.

MIAMEX 28: Output error message

```
xcall MIAMEX, MX_PRINTERR, errnum
```

This function outputs the standard error message associated with the last processed error. It is generally used within LIT programs to display standard error messages. *errnum* is a floating point which will receive the error number.

See ERRMSG for a more powerful and flexible method of handling error messages.

MIAMEX 29: Output rename error message

```
xcall MIAMEX, MX_RENAMERR
```

This function displays a specialized error message of the type associated with a failure to rename a file. It is probably only used with the RENAME.LIT program.

MIAMEX 30: Get queue block contents

```
xcall MIAMEX, MX_GETQUEUE, blkno, qblock
```

This function retrieves the contents of the queue block specified by the numeric parameter *blkno*, returning it in the unformatted variable *qblock*. *qblock* should be mapped as:

```
MAP1 Q'BLOCK                ! Queue data block
   MAP2 QNEXT,b,2           ! Pointer to next queue block
   MAP2 QTYPE,b,2           ! Queue block type (see table)
   MAP2 QOWNER,b,2          ! Owner of queue block
   MAP2 QJUNK,b,2           ! Currently unused
   MAP2 QDATA,x,32          ! 32 bytes of data
   MAP2 QD'FLOCK,x,@QDATA   ! Format of FLOCK record
      MAP3 ACTION,b,2       ! Action
      MAP3 MODE,b,2         ! Mode
      MAP3 RECORD,b,4       ! Record number
      MAP3 CHANNEL,b,4      ! Channel number [12] was 2
   MAP2 QD'XLOCK,x,@QDATA   ! Format of XLOCK record
      MAP3 LOCK1,i,4        ! Primary lock code
      MAP3 LOCK2,i,4        ! Secondary lock code
      MAP3 LOCK3,b,2        ! Lock class: 0=xlock
```

Queue block #0 is a control record of a different layout, which should not be accessed.

Each of the data queue blocks shares the same first four fields, followed by a 32 byte area whose format depends on the block type (*qtype*):

Value	Queue block type
0	Unused block

Value	Queue block type
3	FLOCK – normal
4	XLOCK
6	FLOCK – inactive
7	ZLOCK (proprietary format)
8	RLOCK (proprietary format)
9	FLOCK – pending

The number of queue blocks is set by the QUEUE= statement in the MIAME.INI file.

To scan through all of the active queue blocks, start at *blkno* 2 and then proceed to set each new *blkno* to the *qnext* field, until it becomes zero.

MIAMEX 31: Get environment variable

```
xcall MIAMEX, MX_GETENV, envvar, value
```

This function retrieves the definition of the environment variable whose name is specified in the string parameter *envvar*. If the variable is defined, its value is returned in the string variable *value*. There is no particular length limit to either the name or the value, but the value will be truncated to fit the variable specified.

Environment variables may be defined in various ways depending on the host operating system. Under Windows, you can use the Control Panel "System" applet, or the AUTOEXEC.BAT file. Under UNIX, they are set by shell commands, e.g. "MIAME=/vm/miame". The set command will typically display the current definitions.

In addition to system defined environment variables, A-Shell understands a few special variables which do not have to be set externally. One is "**%CurrentDirectory%**", which will return the current working directory in the host operating system. Two others, which are only available under Windows, are "**%WindowsDirectory%**" and "**%SystemDirectory%**", which return the directory spec of the "Windows" and "Windows System" directories. Another two are set by A-Shell when it launches: **%MIAME%** is the directory where the miame.ini file is, and **%MIAMEFILE%** is the full path spec of the miame.ini file.

MIAMEX 32: Set default file extension

```
xcall MIAMEX, MX_SETEXT, ext
```

This function may be used to set the default file extension for a subsequent file open operation.

MIAMEX 34: Get ISAM version and serial

```
xcall MIAMEX, MX_GETCISAM, fspec, version, serial
```

This function returns the version number and serial number of the ISAM or ODBC library associated with the specified *fspec*. *version* and *serial* should be mapped as strings of 33+ bytes. It is mainly used within ISMUTL.

ISAMPLUS support requires an add-on license to A-Shell. If not licensed, the *serial* string will be returned as "<not licensed>".

MIAMEX 36: Get AMOS hash mode status

```
xcall MIAMEX, MX_GETHASH, status
```

This function returns a flag indicating whether we are in AMOS hash compatibility mode. In this mode, the hash algorithm (used by DIR/H) treats LF line terminators are treated the same as CRLF pairs, so that normal text files will have the same hash code under UNIX, which typically uses LF line terminators, as they do under AMOS, which typically uses CRLF terminators.

status (F,6) returns 0 if not in AMOS hash compatibility mode, else non-zero.

MIAMEX 37: Set/reset AMOS hash mode

```
xcall MIAMEX, MX_SETHASH, flag
```

This function allows you to set or reset the AMOS hash compatibility mode.

status (numeric) should be set to 0 to reset (clear) AMOS hash compatibility mode, else non-zero to set it.

See MIAMEX 36: Get AMOS hash mode status for further notes on AMOS hash compatibility mode. Also note that the SET.LIT command allows you to query and set/reset this option.

MIAMEX 38: Create path

```
xcall MIAMEX, MX_MKPATH, dirspec, status
```

This function creates a new native operating system directory.

dirspec is the directory name to create.

status (F,6) will return 0 for success, else an error number.

This function may (and should) be called for any operating system.

MIAMEX 97: Make directory will be invoked automatically if Function 38 is requested under Windows. Therefore, it is best to always use Function 38 in your application and let A-Shell decide when to convert it to Function 97.

MIAMEX 39: Delete path

```
xcall MIAMEX, MX_RMPATH, dirspec, status
```

This function deletes the specified native operating system directory.

dirspec is the directory name to delete, e.g. “C:\VM\JUNK” or “/vm/junk”.

status (F,6) will return 0 for success, else an error number. You may use MIAMEX Function 86 to translate the error number to a text message.

MIAMEX 41: Zap jobtbl and queue blocks

```
xcall MIAMEX, MX_ZAPQUEUE, jobno
```

This function removes the jobtbl record and any associated queue blocks for the specified job #. Needless to say, this will not be appreciated if the user is actively running, and should only be used in extreme circumstances or when you are sure the user is no longer active.

jobno is the job number of the user to zap.

This function is used by KILL, QUTL and SYSTAT/K/Z commands. The last of these, SYSTAT/K/Z, is the preferred and best way to remove resources left over from a job that has aborted without cleaning itself up.

MIAMEX 50: Get LOKSER status

```
xcall MIAMEX, MX_GETLOKSER, lokflag
```

This function returns a flag indicating if LOKSER is active or not.

lokflag (F,6) will be returned as 0 if LOKSER is not active. A non-zero value indicates that LOKSER is active.

MIAMEX 51: Set LOKSER status

```
xcall MIAMEX, MX_SETLOKSER, lokflag
```

This function allows you to turn LOKSER on or off.

lokflag (numeric) must be set to 0 to turn LOKSER off, or non-zero to turn it on.

You may also use SET.LIT to query or change the LOKSER status.

MIAMEX 52: Get CMD file input status

```
xcall MIAMEX, MX_GETCMDINP, cmdinp
```

This function returns a flag indicating whether command file input mode is active (for all subsequent INFLD operations). See MIAMEX 53: Set CMD file input status for more details.

cmdinp (numeric) will be returned as 0 to indicate that CMD file input mode is not on; non-zero indicates that it is on.

MIAMEX 53: Set CMD file input status

```
xcall MIAMEX, MX_SETCMDINP, cmdinp
```

This function allows you to turn command file input mode on and off for subsequent INFLD.SBR (or INPUT.SBR) calls. In the normal case, BASIC INPUT and monitor level input operations will read from an active command file, but image mode input, such as XCALL INFLD or XCALL INPUT, will not. However, you can override this and force INPUT.SBR and INFLD.SBR to read from a command file, either by passing the CMDINP argument to INFLD, or by adding the "p" character to the SBR=INFDEF: string (in the MIAME.INI), or by using SET CMDINP from the dot prompt, or by setting this flag.

cmdinp (numeric) should be set to 0 to turn command file input mode off, or 1 to turn it on.

MIAMEX 54: Get signal received status

```
xcall MIAMEX, MX_GETSIG, sigmask
```

(UNIX only) This function returns a bitmap indicating which signals have been received since the last time this function was called.

sigmask (F,6) will be returned with the following bits set or cleared:

Value	Name	Meaning
1	SIGINT	Control-C
2	SIGCHLD	Child process terminated
4	SIGUSR1	Receipt of ITC or IJC message
8	SIGUSR2	PolyShell swap operation
16	SIGHUP	Hangup (telnet or terminal session disconnected)
32	SIGKILL	Kill (cannot be trapped so will never be seen)
64	SIGTSTP	Background task waiting for terminal context to perform input operation
128	SIGALRM	Alarm signal (used by sleep timers and WAKNO.SBR)
256	SIGTERM	Default kill signal
512	SIGPIPE	Broken pipe or socket connect (other end has terminated)

MIAMEX 55: Clear signal received status

```
xcall MIAMEX, MX_CLRSIG, sigmask
```

(UNIX only) This function allows you to clear specific bits in the signal received mask. This would be useful in a situation where you wanted to check whether a certain signal was received during a certain time frame. (First you would clear the bit for that signal, then later you would check, using MIAMEX Function 54, if the bit was set.)

sigmask (numeric) should be set to the sum of the signals from the table above (see MIAMEX 54: Get signal received status) that you want cleared. Use –1 to clear all of the signal flags. 0 does nothing.

SUBMIT/W uses this function, along with function 54. First it clears the SIGCHLD signal flag (2), then submits the task, then sleeps, checking every second to see if the SIGCHLD signal flag has been set, which it will be when the submitted task terminates.

MIAMEX 57: Send signal to another process

```
xcall MIAMEX, MX_KILL, pid, signal, status
```

(UNIX only) This function allows you to send any UNIX signal to another process (provided you have sufficient privileges – see note below). It provides essentially the same functionality as the UNIX kill utility (which, despite its name, allows sending any signal).

Parameters

pid (numeric)
is the process ID number of the target process. (You can get this information from SYSTAT, or perhaps by reading the jobtbl directly.)

signal (numeric)
is the signal to send, from the table below. Note that the actual signal numbers associated with the names are not guaranteed to be uniform across versions of UNIX, although these (taken from Linux) are fairly standard. Signals that have no plausible use in applications have been omitted. (Also, note that these numbers are not related at all to the numbers in the SIGMASK used in functions 55 and 56.)

Value	Name	Meaning
0	None	Used to determine if target PID exists
1	SIGHUP	Hangup (telnet or terminal session disconnected)
2	SIGINT	Control-C
3	SIGQUIT	Abort foreground processes, without core dump
6	SIGABRT	Force abort; similar to SIGQUIT
9	SIGKILL	Kill (cannot be trapped)
10	SIGUSR1	User signal (used by ITC, IJC messages)
12	SIGUSR2	User signal (used by PolyShell)
14	SIGALRM	Alarm signal (used by sleep timers and WAKNO.SBR)
15	SIGTERM	Default kill signal; generates error 251

Signal 0 is particularly useful for determining if a target process still exists.

status (numeric variable)
returns 0 to indicate success.

In order to deliver a signal to another process, it must be logged in as the same effective user as the sender, or the sender must be root. One way to ensure this is to use the "setuid" bit on the ashell executable to make all A-Shell users the same effective user. Refer to the *A-Shell Setup Guide* (Installation) for further information on that subject.

MIAMEX 59: Get OPTIONS flags

Documentation update July 2004; see "Update Notes" under MIAMEX 60

```
xcall MIAMEX, MX_GETOPTIONS, options
```

This function returns a bitmap field indicating which OPTIONS have been set. The normal way these options are set is via the OPTIONS= parameter in MIAME.INI (see "Miame.ini name," in table below), or by SET.LIT, or by MIAMEX Function 60.

options (F,6) will return a bitmap field indicating the currently set options, from the table below.

Value	Miame.ini name	Symbol	Notes
1	CRNL	GOP_CRNL	Force CRNL line terminators
2	EXTFIO	GOP_EXTFIO	Extended file I/O options
4	AS400	GOP_AS400	AS400 telnet mode (strip 8s and 11s)
8	LATIN1	GOP_LATIN1	Latin1 character set
16	<na>	GOP_OEM	Set internally for OEM fonts
32	EFFUSR	GOP_EFFUSR	Use effective rather than login user
64	AMOS_ RUNSBR	GOP_AMOS-RUNSBR	Execute AMOS.SBR in-process
128	BRKALC	GOP_BRKALC	Prefill random file allocations with]]]
256	HEXDEC	GOP_HEXDEC	"Hexadecade" dates
512	NOLEADFF	GOP_NO LEADFF	Strip leading formfeeds from print files
1024	NOAUTOXLT	GOP_NOAUTO-XLT	Defeat certain auto F-key translations
2048	NTTS	GOP_NTTS	Force Windows Terminal Server mode
4096	FPROUND	GOP_FPROUND	Round floating point results to 48 bits
8192	FIELDEMU	GOP_FLDEMU	Emulate field terminal (Windows)
16384	EXITWAIT	GOP_EXITWAIT	Forces wait for confirmation before closing session launched with –e
32768	ABS LOOKUP	GOP_ABS-LOOKUP	Return absolute value in LOOKUP
65536	QCLOSE	GOP_QCLOSE	Close qflock between accesses
131072	QBUFFER	GOP_QBUFFER	Normal qflock buffering
262144	PRTCOPIES	GOP_LOCAL-COPIES	Implement multiple spool copies via multiple submissions
524288	SHLPATH	GOP_SHLPATH	Special search path for SHL
1048576	AUTOCCON	GOP_AUTO-CCON	Auto-enable ^C at start of each program
2097152	RAWTABS	GOP_RAWTABS	Raw tab output to terminal
4194304	XABORT	GOP_XABORT	Allow X out of window

Value	Miame.ini name	Symbol	Notes
8388608	NOXABORT	GOP_NO-XABORT	Disable OK in X abort dialog
16777216	STRICT	GOP_STRICT	Strict AMOS compatibility
33554432	NUMPAD_ COMMA	GOP_NUMPAD_ COMMA	Treat decimal point on Windows numpad as comma
67108864	AUTOX	GOP_AUTOX	Allow auto expansion of ISAM and memo files
134217728	ISAM_IDXLOK	GOP_IDXLOK	Lock entire ISAM IDX during update
268435456	MMAPTIME	GOP_MMAPTIME	Force update of modify time of mmaped files every minute
536870912	AUTO_ MEMOPEN	GOP_AUTO_-MEMOPEN	Check user memory when opening file; open in memory if present

MIAMEX 60: Set OPTIONS flags

```
xcall MIAMEX, MX_SETOPTIONS, options
```

This function allows you to set OPTIONS flags on the fly.

options (numeric) may be set to any combination of the option flags in the above table.

Update Notes: Build 875, 11 March 2004

The MIAMEX functions to retrieve and set *options* values have been enhanced to support a second *options* parameter (since we have run out of space to fit all the options into one variable.) The new syntax:

Retrieve options:

```
xcall MIAMEX, MX_GETOPTIONS, options {,options2}
```

Set options:

```
xcall MIAMEX, MX_SETOPTIONS, options {,options2}
```

Note that when setting *options2* values, you can't avoid also setting the *options* values. So you should always first retrieve the current options with MIAMEX,59, then set or clear the desired flags in the *options* and *options2* variables, then use them to update the options in MIAMEX,60. The current options flags are defined in **Error! Reference source not found.** as GOP_xxxxx and GOP2_xxxxx. (The GOP_xxx flags are reproduced in the table above, and the GOP2_xxx flags in the table below):

Value	Miame.ini name	Symbol	Notes
1		GOP2_NOPSDLG	Disable the print screen (^P) dialog and just print directly (see PRINTER command in miame.ini)

Value	Miame.ini name	Symbol	Notes
2	NODELSYS	GOP2_NODELSYS	Don't delete the JOBTBL.SYS and QFLOCK.SYS files on exit, ever. (Otherwise they are delete if no other users are in A-Shell.)
8	GUI_SPC_IND	GOP2_GUISPC-INDENT	Activate "intelligent" GUI indenting (see Proportional Fonts in Dev. Guide)
16	AUTOTPRINT	GOP2_AUTOTPRINT	Treat PRINT statements as TPRINT
256	EFS_OUT_-AMOS	GOP2_EFS_OUT_-AMOS	Auto encrypt output files opened with AMOS specs (requires EFS add-on)
512	EFS_OUT_-HOST	GOP2_EFS_OUT_-HOST	Auto encrypt output files opened with native/host specs (requires EFS)
1024	EFS_ALLO-CATE	GOP2_EFS_-ALLOCATE	Auto encrypt on allocate (requires EFS)
2048	EFS_ALC-INDEX	GOP2_EFS_-ALCINDEX	Auto encrypt on ALLOCATE'INDEXED (requires EFS)
4096	INI_AV	GOP2_INI_AV	VUE looks for ini.av instead of ini.vue
8192	SBX_RUNDIR	GOP2_RUNDIR	Search for SBX in same dir as RUN first
16384		GOP2_NOSPOOL	Display spooling.
32768		GOP2_NOCAPTURE	Disable screen capture (^P)
65536		GOP2_LONGER	Use long (10.3) directory format (DIR/L)
131072		GOP2_MSYNC_MAP	Sync entire map on write to memory mapped file.
262144		GOP2_MSYNC_PAGE	Sync page on write to memory mapped file
1048576	SEQLOK	GOP2_SEQLOK	Sequential file locking (UNIX)

MIAMEX 61: Get TRACE flags

```
xcall MIAMEX, MX_GETTRACE, trflags
```

This function retrieves the current set of trace options.

trflags (F,6) will be set to indicate the current trace options from the table below:

Value	Miame.ini name	Symbol	Notes
1	AMSORT	TROP_AMSORT	Display sorting trace dialog
2	SYSERR	TROP_SYSERR	Display system error dialog
4	SQL	TROP_SQL	SQL tracing
8	LP	TROP_LP	Line printer tracing display
16	SIGNAL	TROP_SIGNAL	Display received signals

Value	Miame.ini name	Symbol	Notes
32	SIGHUP	TROP_SIGHUP	Log hang-ups
64	LOCKS	TROP_LOCKS	Display lock trace
128	LOG	TROP_LOG	Log various details to ashlog
256	JOBS	TROP_JOBS	Log job info to ashlog
512	QOPEN	TROP_QOPEN	Log qflock opening to ashlog
1024	GDIPRT	TROP_GDIPRT	Display GDI print directives
2048	FOPENS	TROP_FOPENS	Log file opens to ashlog
4096	XCALL	TROP_XCALL	Log XCALLs to ashlog
8192	AMOS	TROP_AMOS	Log xcall AMOS ops to ashlog
16384	DEBUG	TROP_DEBUG	Log maximum details to ashlog
32768	INOUT	TROP_INOUT	Log ins & outs to ashlog
65536	BASERR	TROP_BASERR	Log BASIC errors to ashlog
131072	ISAM	TROP_ISAM	Log ISAM ops to ashlog
262144	USRMEM	TROP_USRMEM	Log user memory ops to ashlog
524288	MALLOC	TROP_MALLOC	Log memory allocations to ashlog
1048576	RW	TROP_RW	Log reads and writes to ashlog

MIAMEX 62: Set TRACE flags

```
xcall MIAMEX, MX_SETTRACE, trflags
```

This function allows you to set tracing options on the fly.

trflags (numeric) may be set to any combination of the tracing flags in the table above.

Tracing flags may also be set with the TRACE= statement in the MIAME.INI or with the SET TRACE command. See the *A-Shell Setup Guide* for the MIAME.INI settings for more details about these options.

MIAMEX 63: Load PFK file

```
xcall MIAMEX, MX_SETPFK, pfkspec {,status)
```

This function loads a private (PFK-style) function key translation module into memory. Unlike the IFX and VUX translation tables, which are named after the terminal driver and loaded by default, a PFK-style translation table may have any name (with a PFK extension). When loaded, such a translation table will override any existing translations in the IFX file. Thus, they are handy for loading translations that are specific to a particular program or perhaps even a particular user.

pfkspec (numeric) must contain the file specification of a translation table created by FIXTRN which has a PFK extension (e.g. “MYDIR:ABCDE.PFK”).

status (numeric variable) will be set to 0 on success, or 1 if the file was not found

Use FIXTRN <name>.PFK to create a PFK style translation table. You can also use LOAD <name>.PFK to load it into memory, and DEL to delete it. To delete it within a program, use MIAMEX 109: Delete module from memory.

MIAMEX 64: Display loaded PFK filename

```
xcall MIAMEX, MX_GETPFK, pfkspec
```

This function returns the name of the currently loaded PFK translation file, if any.

pfkspec (string variable) will be set to the filespec of the current PFK file.

MIAMEX 65: Manage screen picture

```
xcall MIAMEX, MX_SCRNPIC, flags
```

This function invokes the screen picture routine to print or capture a picture of the current screen. This is the same routine that is called when you hit Control-P in most input fields, except that here you have the ability to eliminate the pop-up dialog.

flags (numeric) may be any combination of the following:

Value	Meaning
1	Append screen picture to current picture file, if any. (Otherwise overwrite it.)
2	Print the picture file to the default screen picture printer; else just create the print file (in <jobnam>.BUF).
4	Display the screen picture dialog, otherwise just do the operation silently.
8	Delete the picture file after printing.
16	Same as option 4 (display dialog) except that it may be overridden
64	Strips trailing blanks from each line of the output file

The default screen picture printer may be set via the PRINTER=p1,p2{,NODLG} statement in the MIAME.INI file. The first printer(p1), is the default printer for other print operations, and the second printer, if specified (p2) is the default screen picture printer. If the third parameter is specified as “NODLG”, then screen pictures that are requested via the Control-P command will print without displaying the dialog.

MIAMEX 66: Rebuild queue

```
xcall MIAMEX, MX_QRBLD
```

This function forces the qflock.sys file to be rebuilt. This will be done automatically if any broken links are detected, so the possibility of invoking it manually is of only academic interest.

MIAMEX 67: Lock queue

```
xcall MIAMEX, MX_QLOCK
```

This function locks the qflock.sys file. It should be called before trying to read the qflock.sys file, and should then be followed by MIAMEX 68: Unlock queue to unlock it.

MIAMEX 68: Unlock queue

```
xcall MIAMEX, MX_QUNLOK
```

This function removes the lock on the qflock.sys file which was placed by a prior call to MIAMEX 67: Lock queue.

MIAMEX 69: Get UMASK

```
xcall MIAMEX, MX_GETUMSK, umask
```

(UNIX only) This function retrieves the current UMASK setting.

umask (numeric variable) will return with the current value of the UMASK, treating it as if it were a decimal number.

For example, if the UMASK is 111 (which is actually an octal pattern), the returned *umask* value will be 111 (decimal).

MIAMEX 70: Set UMASK

```
xcall MIAMEX, MX_SETUMSK, umask
```

(UNIX only) This function establishes a new UMASK setting.

umask (numeric) should be set to the 3-digit decimal number, which if interpreted as octal, represents the desired UMASK setting. For example, to set the UMASK to 117, simply set *umask*=117. (In other words, you can ignore the fact that the UMASK is actually an octal representation of a bit pattern.)

A-Shell's default UMASK is 111. If you set *umask*=000, A-Shell will instead use the mask set by the shell. Otherwise it will use the specified mask.

Note that the bits in the umask clear the corresponding bits in the file creation mask, which starts at 666 (rw-rw-rw-). So setting UMASK=111 will result in no change to the rw-rw-rw- mask. But UMASK=137 would result in rw-r---- (rw by the file owner, read only by members of the group, otherwise no access).

You may also set the UMASK in the MIAME.INI file, using UMASK=xxx.

MIAMEX 71: Windows menu operations

This is identical to the AUI Menu class, which should be used instead, and which see for further information.

MIAMEX 72: Print msg from msg file

Documentation update November 04; added new parameter "default message" in build 904.2, 18 October 2004. See description below.

```
xcall MIAMEX, MX_LIGMSG, cat, msgno, row, col, var, msgfile, arg,
      "Default Message"
```

This function is mainly used within LIT utilities to display language-independent messages, although you can add use it to reference your own (properly formatted) message files as well. A "properly formatted" message file consists of lines in the following format:

```
<cat>,<msgno>,<message>{,<arg>}
<cat>,<msgno>,<message>{,<arg>}
etc
```

<cat> and <msgno> are typically formatted as three digit numbers, as in the example below. Any message may optionally be followed by a comma a numeric value which can optionally be returned to the calling program for whatever purpose. Comments are typically inserted by using <cat> and/or <msgno> values of 000. The lines of the file must be sorted, first by <cat> and then by <msgno>. As an example, here is an excerpt from the LITMSG.USA file:

```
000,000,#
000,000,# A-Shell LIT command message table
000,000,# Language: English (US Only)
000,000,#
000,000,# Following 000,### messages are generic lit/cmdlin messages
000,000,#           001,### messages are from about.lit
000,000,#           002,### message are from telset.lit, etc.
000,001,?Exceeded maximum number of devices
000,002,?Cannot read
000,003,greater than
000,004,^ Specification error
000,005,?Device not found or mounted
000,006,file
000,007,disk block
000,008,Total of
000,009,%No files transferred
000,010,transferred
000,011,%No files deleted
000,012,deleted
001,000,# ABOUT.LIT messages...
001,001,Serial #
001,002,Licensed to:
001,003,Nodes Licensed:
001,004,Nodes in Use:
001,005,Jobs in Use:
```

cat (numeric) is the message category number (corresponding to <cat> in the example above). In the case of the LITMSG.xxx files, each LIT utility would have its own *cat* value.

msgno (numeric) is the message number within the *cat* category (corresponding to <msgno> in the example above).

row and *col* (numeric), may be optionally specified to have the message displayed at the given row and column position. If either is zero, or they are omitted, then the message will be displayed at the current cursor position. If *row* is negative, then the message is not displayed but instead only returned in the *var* parameter.

var (string variable) will return the text of the message if specified.

msgfile (string) may optionally give the complete filespec of the message file. If omitted or blank, the default is "DSK0:LITMSG.xxx[1,4]" where xxx corresponds to the current language (as defined in the LDF or language definition file specified in the LANGUAGE statement of the MIAME.INI).

arg (numeric variable) if specified, will return the optional <arg> field of the current message, if any.

"default parameter" both makes it obvious what the expected text of the message is, as well as eliminating the problem of what to do if the message wasn't defined.

MIAMEX 73: Read job control block

xcall MIAMEX, MX_READJCB, jobno, buffer {,lokflg}

This function reads a job control block or record from JOBTBL.SYS.

jobno (numeric) is the job number of the block to read. (This is equivalent to the record number, starting at 1 for the first job.) *jobno* 0 refers to the control record, which should not be accessed.

buffer is the record buffer to receive the data. It should be mapped as:

```
MAP1 JCB'REC                            ! JCB Record
     MAP2 JCB'PID,i,4                   ! Process id
     MAP2 JCB'PPID,i,4                  ! Parent's Process id
     MAP2 JCB'TIMELOGGEDIN,i,4          ! Time when we logged in
     MAP2 JCB'TOTREADS,i,4              ! Total disk reads
     MAP2 JCB'TOTWRITES,i,4             ! Total writes
     MAP2 JCB'TOTQLOCKS,i,4             ! Total qlocks
     MAP2 JCB'TOTINSTR,i,4              ! Total BASIC instructions
     MAP2 JCB'TOTCMDS,i,4               ! Total LIT commands
     MAP2 JCB'TOTKEYS,i,4               ! Total keystrokes
     MAP2 JCB'JOBTYP,b,2                ! Job type (see JOBTYP'xxx)
     MAP2 JCB'JOBSTS,b,2                ! Job status (see JOBSTS'xxx)
     MAP2 JCB'JOBNAM,s,8                ! Job name
     MAP2 JCB'TRMTDV,s,8                ! Terminal driver
     MAP2 JCB'PRGNAM,s,8                ! Program name
     MAP2 JCB'DRIVE,s,8                 ! DSKn:
     MAP2 JCB'PROJ,b,1                  ! Project
     MAP2 JCB'PRGN,b,1                  ! Programmer #
     MAP2 JCB'JOBLVL,b,1                ! Job experience level
     MAP2 JCB'JOBCLS,b,1                ! Job class
     MAP2 JCB'CONDEV,s,20               ! Machine or device
     MAP2 JCB'USRNAM,s,20               ! User login name
     MAP2 JCB'MEMK,b,2                  ! Memory in K
     MAP2 JCB'JOBNO,b,2                 ! Our job # (1 is first)
     MAP2 JCB'IJCCMD,b,1                ! Inter-job Comm command
     MAP2 JCB'IJCSTS,b,1                ! Inter-job Comm status
```

```
    MAP2 JCB'IJCMSG,b,1                 ! Inter-job Comm msg type/fmt
    MAP2 JCB'IJCBUF,s,110               ! Inter-job Comm msg buf
    MAP2 JCB'IP(4),b,1                  ! IP address
    MAP2 JCB'LASTOP,b,1                 ! Last BASIC opcode
    MAP2 JCB'LASTLNO,b,2                ! Last BASIC line #
    MAP2 JCB'LASTSBR                    ! Last sbr (rad50)
        MAP3 JCB'LASTSBR1,b,2
        MAP3 JCB'LASTSBR2,b,2
    MAP2 JCB'JOBFIL,x,12                ! Future expansion
```

The *jcb'jobtyp* field values are defined as follows:

```
MAP1 JOBTYP'SYMBOLS                     ! see JCB'JOBTYP
    MAP2 JOBTYP'FREE,b,2,0              ! free rec
    MAP2 JOBTYP'ACTIVE,b,2,1            ! active job
    MAP2 JOBTYP'ALLOC,b,2,2             ! allocated via trmdef
    MAP2 JOBTYP'HEX,b,2,16              ! hex mode
    MAP2 JOBTYP'SLAVE,b,2,512           ! background task
    MAP2 JOBTYP'PSHELL,b,2,1024         ! pshell controller
    MAP2 JOBTYP'DISCON,b,2,2048         ! job disconnected
    MAP2 JOBTYP'EOF,b,2,32768           ! eof
```

The *jcb'jobsts* field values are defined as follows:

```
MAP1 JOBSTS'SYMBOLS                     ! see JCB'JOBSTS
    MAP2 JOBSTS'MON,b,2,1               ! job at monitor level
    MAP2 JOBSTS'SUS,b,2,2               ! job suspended
```

lokflg (numeric) may optionally be set to 1 to lock the record prior to reading it (leaving it locked thereafter). To unlock it you would then need to write the record back, using Function 74, and also specifying the same *lokflg*=1.

> To scan all of the job table records, start at JOBNO=1 and keep incrementing it, skipping records with `JCB'JOBTYP = JOBTYP'FREE`, and quitting when you hit a record with `JCB'JOBTYP = JOBTYP'EOF`.

MIAMEX 74: Write job control block

`xcall MIAMEX, MX_WRITEJCB, jobno, buffer {,lokflg}`

This function is identical to the previous one (function 73), except that it writes a jobtbl record instead of reading it. Needless to say, it should used with extreme caution, since writing invalid information to JOBTBL.SYS could cause undesirable results.

The parameters are the same as for function 73.

MIAMEX 75: Get system time

`xcall MIAMEX, MX_GETTIME, secs`

This function returns the number of seconds since 00:00:00 January 1, 1970, Coordinated Universal Time (aka the "Epoch").

secs (numeric variable) will receives the number of seconds elapsed since the "Epoch". Since as of 2003 this number is well over 1 billion seconds, you would be well advised to map *secs* as a 4 or 5 byte binary or a floating point.

> The TIME system variable in BASIC returns the number of seconds since midnight.

MIAMEX 76: Start telnet server

```
xcall MIAMEX, MX_TELSER, port, status {,wait}
```

(Windows only) This function allows you to turn your current A-Shell/Windows client into a telnet server for a single remote client. Your session will continue to run normally until a telnet client connects; at that point, your window will be hidden and all terminal I/O redirected to the remote telnet client.

Parameters

port (numeric)
specifies the port number to accept the incoming connection request on.

status (F,6)
returns a code indicating the success of the operation, from the table below:

Value	Meaning
0	Success
-1	Unable to load Winsock library
-2	Winsock 1.1+ not present
-3	Unable to create listening socket
-4	User abort (^C while waiting for connection)

Wait (numeric)
may be used to specify options (combine zero or more from the following table):

Value	Meaning
1	Minimize window immediately and wait for connection before returning from subroutine. (Otherwise it returns immediately with the wait operation waiting in background for the connection.)
2	Launch a new instance to wait for a subsequent connection upon accepting the first connection
4	Make waiting window totally invisible

You can restore visibility to an invisible window waiting for a connection by using SEND.LIT or SEND.SBR to send it a message consisting of just an exclamation point (!).

MIAMEX 77: Show window

```
xcall MIAMEX, MX_SHOWWINDOW, flags
```

(Windows only) This function redisplays the A-Shell window according to the specified option. (This is directly equivalent to the Windows "ShowWindow()" function).

flags (numeric) must be set to one of the following:

Value	Meaning
0	Hide window (i.e. make it invisible)
1	Show window in normal format (neither minimized nor maximized)
2	Activate window and show it minimized (displayed on the task bar)
3	Activate window and show it maximized
4	Show window in its most recent position but do not activate it
5	Activate and show window in its current size and position
6	Minimize window and activate the next window in the task list
7	Maximize the window
8	Display window in its current size and position, but do not activate it
9	Restore a minimized or maximized window to its original position
10	Display the window in the default state as defined in the shortcut or startup info associated with the application

MIAMEX 78: Get global DO params

```
xcall MIAMEX, MX_GETGDO, status, g0 {,g1 {,g2 {,g3 {,g4 {,g5 {,g6
      {,g7 {,g8 {,g9}}}}}}}}}
```

This function retrieves the currently set global DO parameters. These are parameters that may be set within a program and then retrieved later and used as arguments in a DO file command line.

status (numeric variable) returns 0 for success or –1 to indicate failure.

go thru *g9* (string variables) will receive the current values of the up to 10 global DO file parameters.

See the *A-Shell Command Reference* for information on using global DO parameters in a DO file.

MIAMEX 79: Set global DO params

```
xcall MIAMEX, MX_SETGDO, status, g0 {,g1 {,g2 {,g3 {,g4 {,g5 {,g6
      {,g7 {,g8 {,g9}}}}}}}}}
```

This function may be used to set up to 10 global DO parameters. These are parameters may be retrieved later using Function 78 or used as arguments on a DO file command line. The entire set of up to 10 DO file

parameters is set at once, replacing any prior values. (Even if you specify only *g0*, the prior values of *g1* thru *g9* will be set to null.)

status (numeric variable) returns 0 for success or –1 to indicate failure.

go thru *g9* (string variables) will receive the current values of the up to 10 global DO file parameters.

See the *A-Shell Command Reference* for information on using global DO parameters in a DO file.

MIAMEX 80: Get XCALL param info

```
xcall MIAMEX, MX_XCBINFOX, xcbadr, xcbcnt, xcbstructx
```

This function is needed at the start of an **Error! Reference source not found.** subroutine to retrieve information about the parameters which were passed to the subroutine. Because every subroutine needs this information, the XCALL.BSI file which is included by every subroutine includes a call to this function.

Parameters

xcbadr (numeric)
must be set to the address of the parameter structure. (This value is passed on the command line to the SBX routine.)

xcbcnt (numeric variable)
returns the number of parameters passed.

xcbstructx
is a structure containing information about the parameters, mapped as follows:

```
MAP1 XCBSTRUCTX
   MAP2 XCBSTRUCT(20)          ! up to 20 parameters
      MAP3 XCB'PTYPE,b,2      ! 0=X, 2=S, 4=F, 6=B, +8=array)
      MAP3 XCB'PSIZE,b,2      ! size of parameter
```

Refer to the documentation on writing subroutines in the *A-Shell Development Guide* for more information on Functions 80, 81, 82 and 89.

MIAMEX 81: Get/set params in SBX

```
xcall MIAMEX, MX_XCBDATAX, xcbadr, op, parmno, var
```

This function is needed to pass parameter data between a calling program and an external **Error! Reference source not found.** routine.

Parameters

xcbadr (numeric)
must be set to the address of the parameter structure. (This value is passed on the command line to the SBX routine.)

op (numeric)

must be set to 0 to retrieve a parameter or 1 to pass it back. (For convenience, these values are mapped in the XCALL.BSI file as *xcbget* and *xcbput*, respectively.)

parmno (numeric)
is the number of the parameter to receive or send (starting from 1).

var
is the actual variable to receive or send. It should be mapped in a way that is "reasonably compatible" with the actual type and size passed to the subroutine, although conversions and truncations will be applied automatically if necessary.

Refer to the documentation on writing subroutines in the *A-Shell Development Guide* for more information on Functions 80, 81, 82 and 89.

MIAMEX 82: Exit SBX

```
xcall MIAMEX, MX_EXITSBXX
```

This function may be used to abort from an **Error! Reference source not found.** subroutine directly to the dot prompt. This is necessary, since merely ending an SBX routine (with the END statement) will return to the calling program.

Refer to the documentation on writing subroutines in the *A-Shell Development Guide* for more information on Functions 80, 81, 82 and 89.

MIAMEX 83: Get FKEYWAIT time

```
xcall MIAMEX, MX_GETFKW, waitms
```

(UNIX only) This function retrieves the current FKEYWAIT (function key sequence completion wait) value for this terminal.

waitms (numeric) will receive the wait time, in milliseconds.

MIAMEX 84: Set FKEYWAIT time

```
xcall MIAMEX, MX_SETFKW, waitms
```

(UNIX only) This function sets the current FKEYWAIT (function key sequence completion wait) value for this terminal.

waitms (numeric) should be set to the desired wait time, in milliseconds.

See the discussion of FKEYWAIT in the MIAME.INI section of the *A-Shell Setup Guide*.

MIAMEX 85: Get or set window title

```
xcall MIAMEX, MX_TITLE, op, title
```

This function gets or sets the title string which will be displayed in the Windows title bar or applicable terminal emulator.

op (numeric) should be set to 0 to retrieve the current title or 1 to set it.

title (string variable) is the title string, either returned for *op* 0 or to set for *op* 1.

The *title* string may contain any of the following special macros:

Title	Meaning
$TS	Display title on top status line rather than on title bar of window. Valid only at the start of the TITLE string.
$NC	Name of current program or LIT command
$ND	Name of current disk device, e.g. “DSK0”
$NJ	Name of current job
$NP	Name of current program (RUN only)
$PA	Platform A-Shell compiled for, e.g. “Windows/32”
$PN	Current PPN, e.g. “[7,6]”
$VA	A-Shell version, e.g. “4.7(825)”
$VP	Version of current program

This function can also be performed using the SET.LIT TITLE option. Note that if $NC, $ND, $NP, or $PN is specified, the title display will be automatically updated whenever the corresponding information changes.

MIAMEX 86: Get error message

```
xcall MIAMEX, MX_ERRNOMSG, errno, message
```

This function returns the operating system’s text message associated with the specified operating system error code (aka **errno**).

errno (numeric) is the error number for which you want the message. Note that many MIAMEX functions return operating system errors as negative numbers; these should be converted to absolute value before passing to this function.

message (string variable) will return the error message.

MIAMEX 87: Disable stream buffering

```
xcall MIAMEX, MX_NOBUF, channel
```

(UNIX only) This function will disable the normal stream buffering on output streams (i.e. files open for output, and the console itself, aka *stdout*). By default UNIX systems buffer output streams to minimize the amount of system overhead involved in writing to them. For files, the buffer size may be several K. For the terminal or console, the buffering is generally limited to a single line.

In general this buffering is highly desirable, but there may be cases where it is annoying. One situation occurs when you are treating a physical port as a file, in order to control device connected to that port. In such a case, you may write out a small command to the file/port thinking that the device will see it immediately, but due to the buffering, it would not. A more typical situation involves writing some kind of status information (aka "life signs") to the screen during a lengthy process. Unless each update of the status information ended with a line terminator, it would simply be buffered, with the result that instead of seeing dots or percentage completion indicators steadily appear on the screen, there is a delay, and then it all appears at once. One (but not the only) solution to these problems would be to just disable buffering on the file channel in question.

> *channel* (numeric) is the file channel to disable buffering on. The file must be open for output (or append). For the terminal device, use file channel 0 (which is automatically opened for you at the start of each program).

An alternate approach is to force the buffer to be flushed on demand. For terminal devices, you can do this with TAB(-1,254), or simply by ending a PRINT statement without a semicolon. For file devices, you can use Function 88.

This buffering issue does not affect data files that are open for random I/O, ISAM, etc.

MIAMEX 88: Flush stream buffer

```
xcall MIAMEX, MX_FLUSHBUF, channel
```

(UNIX only) This function is similar to the previous one but instead allows you to flush (or "unbuffer") the data, which has been buffered but not yet written to the specified file channel.

> *channel* (numeric) is the file channel to flush the buffer of. The file must be open for output (or append). For the terminal device, use file channel 0 (which is automatically opened for you at the start of each program).

See MIAMEX 87: Disable stream buffering for more information about buffering.

MIAMEX 89: FLSET

```
xcall MIAMEX, MX_FLSET, ch, status, recvar'adr {,stavar'adr}
```

This function may be used within an **Error! Reference source not found.** routine to link up with the control variables associated with a random or ISAM file that was opened in the calling program.

Parameters

ch
: is the file channel (opened in the parent program) that you want to access.

status (F,6)
returns to a code reflecting the outcome of the operation:

Value	Meaning
-1	Parameter error
0	File CH was not open
1	Success: file was opened for INDEXED access (ISAM 1.0)
2	Success: file was opened for INDEXED'EXCLUSIVE
3	Success: file was opened for RANDOM access
4	Success: file was opened for INPUT
5	Success: file was opened for OUTPUT or APPEND
6	Success: file was opened for INDEXED access (ISAMPLUS)
7	Success: file was opened for INDEXED'EXCLUSIVE (ISAMPLUS)

recvar'adr (X,6)
is an unformatted variable whose origin or location is the same as the F,6 variable which you want to use for specifying the record number for this file. We have to resort to this trick of mapping it as an X type so that we can get the true address of the variable, because otherwise floating point arguments to subroutines are pushed on the stack. So, for example, you should map it like this:

```
MAP1 RECNOX                 ! (specify this to MIAMEX, MX'FLSET)
    MAP2 RECNO,F,6          ! (actual record number control var)
```

or

```
MAP1 RECNO,F,6              ! (specify this to MIAMEX, MX'FLSET)
MAP1 RECNOX,X,6,@RECNO      ! (actual record number control var)
```

In either case, specify the *recnox* variable in the MIAMEX call, but use the *recno* variable to set the record number within the file.

statvar'adr (X,6)
is only used in the case of an ISAMPLUS file, and is used to specify the ISAM status variable that will return the result of each ISAMPLUS operation. It must be mapped and specified to the subroutine using the same trick as for *recvar'adr* above.

Comments

Once you successfully execute the MX_FLSET operation, you can have full access to the RANDOM, ISAM, or ISAMPLUS file within the subroutine.

Refer to the documentation on writing subroutines in the *A-Shell Development Guide* for more information on file handling and other related issues in **Error! Reference source not found.** routines.

MIAMEX 90: Get/set beveling options

```
xcall MIAMEX, MX_BEVEL, op, bevel, linedraw
```

(Windows only) This function allows you to query or set the beveling options, which are otherwise available on the Edit menu or via the SET BEVEL command.

Parameters

op (numeric)
should be set to 0 to query the current options or 1 to set them

bevel (numeric)
indicates the bevel mode:

Value	Meaning
0	Beveling off
1	Beveling set to automatic
2	Beveling set to program control. In this case, to turn beveling on within a program, you can use "PRINT TAB(-10,13); str(FLAGS); chr(127)" where FLAGS are values that control beveling options, with the most useful choices being 0 (off), 255 (on, with no line drawing characters) or 15 (beveling, but use line drawing chars also).

linedraw (numeric)
should be set to 0 for enabled, or 1 for disabled. (The recommendation is to disable line drawing when beveling is active so that lines will be represented entirely by the 3D raised objects.)

MIAMEX 91: Get/set background color

Documentation update June 04; see "Update Notes" below.

```
xcall MIAMEX, MX_SYSBCLR, op, bgcolor
```

(Windows only) This function allows you to assign one of the background color numbers (0-15) to reference the default Windows color for dialog box backgrounds (typically gray). The idea would then be to use `PRINT TAB(-3,BGCOLOR)` to use that color as your background color. The advantage of tying this color number to the Windows system background color instead of just setting it to a specific color, is that then it will automatically change when the Windows desktop color scheme is changed. For example, if the user switches to a "Valentine's Day" scheme, the cool gray Window backgrounds will all change to pinkish ones.

Parameters

op (numeric)
may be specified as 0 to query the current system background color number, or 1 to set it.

bgcolor (numeric variable)
will return the currently defined system background color number (for *op* 0), or establish it (for *op* 1).

You can also set this with SET TERM SYSBCOLOR n.

Update Notes: Build 863.5 - 02 Feb 04

MIAMEX 91 has been enhanced to allow you to associate one of your 16 palette colors with two other Windows system colors: "window color" (typically white) and "text color" (typically black):

```
xcall MIAMEX, MX_SYSBCLR, op, sysbgc {,syswinc{, systxtc}}
```

syswinc (0-7) applies to one of the background colors in the color palette, while *systxtc* (0-7) refers to one of the foreground colors.

The advantage of using these colors is that the user can then adjust your color scheme using the Control Panel desktop applet.

It is preferable to use this method of associating a background color number with the "system gray" color, rather than simply trying to pick a suitable gray from the color palette, because this not only guarantees that you get exactly the right gray, but also allows certain A-Shell routines to optimize themselves by using the system palette. Consequently, the recommendation would be to add code to execute MIAMEX,91 in your startup routine. Or at the very least add a SET TERM SYSBCOLOR ## command to your startup command file.

MIAMEX 92: Enable/disable PolyShell keys

```
xcall MIAMEX, MX_HOTKEY {,enable}

xcall MIAMEX, MX_HOTKEY, hotkey, swapkey
```

(UNIX only) This function allows you to temporarily disable the PolyShell hot or swap keys, or reassign them. The typical motivation for temporarily disabling them is in situations where you are allowing an external process to control the screen (for example, an AutoLog communication session). In such a case, swapping jobs with PolyShell may not be advisable.

Parameters

enable (numeric)
may be set to zero to disable the hot and swap keys, or 1 to re-enable them. (The default, if the parameter is omitted, is 0 to disable.)

hotkey (numeric)
may specify the ASCII key value of the new PolyShell hot key (or 0 to disable it).

swapkey (numeric)
may specify the ASCII key value of the new PolyShell swap key (or 0 to disable it).

MIAMEX 93: Get ABOUT info

```
xcall MIAMEX, MX_ABOUT, prdname, version, serial, coname, key,
      licnodes, phynodes, lognodes, licoptions, expflags, reldate,
      expdate, inifile {,jobtbl'fspec, jobtbl'inode, jobtbl'cdate,
      jobtbl'ctime}
```

This function retrieves several pieces of information relating to the current license and environment. It is mainly used by ABOUT.LIT, but might be useful within in application to check for maintenance expiration, number of licensed users, etc. A quick look at the output of ABOUT.LIT will answer many questions about the parameter output.

Parameters

prdname (string)
will return the A-Shell platform-specific name, e.g. “A-Shell/Linux” or “A-Shell/Windows/32”.

version (string)
will return the A-Shell version string, e.g. “Ver 4.7(825)”.

serial (numeric)
will return the serial number.

coname (string)
will return the company name that the license was issued to.

key (string)
will return the license key (aka PIC code).

licnodes (numeric)
will return the maximum number of nodes licensed.

phynodes (numeric)
will return the current number of physical nodes in use.

lognodes (numeric)
will return the current number of logical nodes in use. (The logical node count should count each job, whereas the physical count may be less if multiple jobs are identified as being connected with the same session or are running in background.)

licoptions (string)
will return a string of descriptive tokens identifying license options, such as “COM-XCALL”, “PolyShell”, etc.

expflags (numeric)
will return 0 to indicate the normal maintenance expiration, or 1 to indicate that the license has a runtime expiration (i.e. will stop running after the expiration date). A 16 or 17 indicates that the expiration date has passed.

reldate (string)
will return the release date of the current executable (dd-mon-yyyy).

expdate (string)
will return the maintenance or runtime expiration date of the license (dd-mon-yyyy).

inifile (string)
will return the fully qualified native filespec of the MIAME.INI file.

jobtbl'fspec (string, optional)
will return the fully qualified native filespec of the jobtbl.sys file.

jobtbl'inode (numeric, optional)
will return the inode of the jobtbl.sys file. (Applies to UNIX only; will be set to 0 under Windows.)

jobtbl'cdate (string, optional)
will return the creation date (or last status change date) of the jobtbl.sys file (dd-mmn-yy) format.

jobtbl'ctime (string, optional)
will return the creation time (or last status change time) of the jobtbl.sys file (hh:mm) format.

Under UNIX, the "c date" will be updated by any change to the privileges or ownership of the file, so it may not reliably indicate the actual creation time.

MIAMEX 94: Get cursor position

```
xcall MIAMEX, MX_GETRC, row, col
```

This function retrieves the current position of the cursor.

row and *col* (numerics) will return the current cursor row and column, respectively.

MIAMEX 95: Display open file dialog

```
xcall MIAMEX, MX_GETOFD, path, filter, title, flags {,defext, file,
      type}
```

(Windows only) This function displays the standard Windows open file (or save file) dialog.

Parameters

path (string)
should be passed as the initial (default) filename or just a directory, if applicable, and will be returned with the complete native filespec of the selected file. You may use AMOS notation, e.g. "SYS:", or "DSK5:[123,4]", or "MYDOCS:SAMPLE.TXT" when specifying the initial file or directory, but it will be returned in native format (so a size of 100 or more is recommended). Note that when multiple files are allowed, they are all concatenated together in this variable, separated by null bytes.

filter (string)
is a set of pairs of strings, concatenated all together using pipes ("|") for delimiters, determining which files will display in the dialog box. The first string in each pair is descriptive, and will display in the "Files of Type" area of the dialog box. The second string in each pair is a wildcard specification for that kind of file. If you have more than one wildcard spec for a particular descriptive name, then separate them with semicolons. For example:

```
FILTER="Data files | *.DAT | Text files | *.TXT; *.LST"
```

Note that the "Files of Type" control is actually a drop-down box and that "Data files" will show as the initial choice, and "Text Files" will be the next choice in the list. If you want them all appear together, then just specify one pair of strings, such as:

```
FILTER="Image Files (*.pcx, *.jpg, *.bmp) | *.pcx; *.jpg; *.bmp"
```

title (string)

may be set to the desired title of the dialog box. The default is “Open”. This is the main difference between the File Open and File Save versions of the dialog box.

flags (numeric)

may be set to any combination of the flags shown on the table below. The table also indicates the Windows name for the flag, which might be of use to programmers who are familiar with the GetOpenFileName and GetSaveFileName functions, or who have access to the Windows API documentation. The symbol names are not defined to A-Shell, but if you use this function, you may want to use correspondingly named variables to make your code understandable. *flags* may be updated by the function on return.

Value	Windows symbol	Meaning
4	OFN_HIDEREADONLY	Hides the read-only checkbox
8	OFN_NOCHANGEDIR	Disable the ability to change the directory
256	OFN_NOVALIDATE	Do not force filename entered to contain only valid chars
512	OFN_ALLOWMULTISELECT	Allow the selection of multiple files. (OFN_EXPLORER flag should be set along with this.) Successive names will appear, null delimited, in the returned PATH variable.
1024	OFN_EXTENSIONDIFFERENT	Set on return if the extension entered differs from *defext*
2048	OFN_PATHMUSTEXIST	Specified directory must exist
4096	OFN_FILEMUSTEXIST	Specified file must exist. (Implies OFN_PATHMUSTEXIST)
8192	OFN_CREATEPROMPT	Prompt for permission to create the file if the file does not already exist
32768	OFN_NOREADONLYRETURN	Set on return if file not read-only, directory writeable
131072	OFN_NONETWORKBUTTON	Disables the network button
524288	OFN_EXPLORER	Forces the dialog box to be in the new Explorer-style. This is the default in most cases, except when OFN_ALLOWMULTISELECT is set.
33554432	OFN_DONTADDTORECENT	(Win2000/XP) Do not add file to the user’s most recently used document list

defext (string)

may specify the default file extension. Do not include the period. If specified and the user types a filename with no extension, this will be appended to the returned PATH and FNAME parameters.

fname (string)

is an optional convenience that will return just the filename and extension (without the rest of the path), saving you from having to parse it out of the PATH string. Note, however, that it is not guaranteed to be in the directory you initially specified.

MIAMEX 96: Shell execute

```
xcall MIAMEX, MX_SHELLEX, status, spec {,action {,parms {,dir
      {,showflags}}}}
```

(Windows only) This function performs the default (open) or specified action on the specified file or URL. Such actions are defined in the Windows Registry by means of the File Types dialog (on the Explorer Tools...Folder Options menu).

Parameters

status (F,6)
will return one of the following:

Value	Meaning		Value	Meaning
0	Success		11	Bad format
-1	Out of system resources		26	Sharing error
2	File not found		27	File association incomplete
3	Path not found		31	No association defined
5	Access denied		32	DLL not found
8	Out of memory			

spec
is the specification (Windows format) of the file/object to open or act on, or the URL to link to. For example, you might specify a XLS file, such as "C:\Windows\Temp\My Spreadsheet.XLS", in which case the presumed action would be to launch the spreadsheet program (e.g. Excel) to open the spreadsheet. For, you might specify a URL, such as "http://www.microsabio.com" in which case the presumed action would be launch the browser and link to the specified web site or HTML document.

action (string)
may specify an action (e.g. "print", "edit", etc.). If omitted, the default action for that file association is performed. Such actions are defined with the file association in the Explorer Tools...Folder Options...File Types dialog.

parms (string)
is an optional list of parameters. This would only be valid for non-document objects, and again would depend on the associations defined for that object.

dir (string)
is an optionally starting directory. This can generally be omitted or specified as "".

showcmd
is an optional numeric parameter specifying the way to show the window launched by the program associated with the object and action. If omitted, the default window size will be used. Note that if you do specify the parameter, a value of zero means to make the window invisible. The values are the same as for MIAMEX 77: Show window. Note that this is only a suggestion to the target application. There is no way to force it to follow your suggestion if it insists otherwise.

Comments

This method is generally preferred over trying to launch an application by its name, since it eliminates three of the biggest problems normally associated with launching local applications: knowing the actual name of the executable, the command line syntax, and directory where it is located. All of these are take care of by the association definition.

Update Notes: Build 934, 17 June 05

MIAMEX,96 (MX_SHLEXC) now supports embedded environment variables (using the %env% notation, e.g. %temp%\test.doc).

MIAMEX 97: Make directory

```
xcall MIAMEX, MX_MKDIR, dirspec, status
```

(Windows only) This function provides a native Windows equivalent to MIAMEX Function 38 to create a directory.

Parameters

dirspec is the directory name to create.

status (numeric) will return 0 for success, else an error number.

Comments

Function 97 will be invoked automatically if Function 38 (MIAMEX 38: Create)is requested under Windows. Therefore, it is best to always use Function 38 in your application and let A-Shell decide when to convert it to Function 97.

MIAMEX 99: Get registry string

```
xcall MIAMEX, MX_GETREG, key, subkey, name, value
```

(Windows only) This function allows you to retrieve values from the system Registry. Also see MIAMEX 138: Registry operations, which offers a more extensive set of registry access capabilities.

Parameters

key

(numeric literal or B,4) must be one of the following specified values:

Key	Meaning
2147483648	HKEY_CLASSES_ROOT
2147483649	HKEY_CURRENT_USER
2147483650	HKEY_LOCAL_MACHINE
2147483651	HKEY_USERS
2147483652	HKEY_PERFORMANCE_DATA

Key	Meaning
2147483653	HKEY_DYN_DATA

subkey (string)
is the sub key within the specified *key* section of the Registry. This may have multiple parts, e.g. `"Software\Microsoft\MediaPlayer\Player\RecentFileList"`.

name (string)
is the name of the particular value to look for within the specified *subkey*. You may specify a blank string to get the default value. (Many *subkey*s have only a default value.) For the *subkey* example given above, the likely names will be `"File0"`, `"File1"`, etc.

value (string)
will return the value of the specified item. It will be truncated if needed to fit in the variable provided.

Comments

This function is only useful for registry items of type string. Also, there is an internal limit of 256 bytes for the value. Even if you specify a larger variable for VALUE, if the actual data exceeds 256 bytes, the operation will fail and nothing will be returned. (This internal limit was 64 prior to the final 4.7 release.)

MIAMEX 100: Play sound

```
xcall MIAMEX, MX_PLAYSOUND, fspec {status {,flags}}
```

(Windows only) This function allows you to play a WAV (sound) file directly. You could also use MIAMEX function 86 to launch the media player to play a specified sound file, but this method avoids the complication of another application and gives you more control.

Parameters

fspec
must contain the file specification (AMOS or Windows syntax) of the WAV file to play (or the "event label" – see *flags* value 65536 below).

status (numeric, optional)
returns 0 (FALSE) if it failed to play, else non-zero (TRUE) for success.

flags (numeric, optional)
may be used to specify flags used by the Windows PlaySound() function:

Value	Meaning
1	Return immediately (while sound still playing). To terminate the playback later, call this function again with *fspec*="".
8	Play sound continuously. Must be used with flag 1 (1+8=9). To terminate the sound later, call with *fspec*="".
16	Do not stop a currently playing sound. (Otherwise a request to play a sound file will terminate any sound file currently playing; this is why setting *fspec* to null terminates the current sound file.)

Value	Meaning
8192	Do not wait if the sound driver is busy.
65536	FSPEC is not interpreted as a file specification but as an “event label” defined in the registry. See HKEY_USERS\.Default \AppEvents \EventLabels. These are the “sound events” that you can associate sound files with using the Control Panel Sound applet (only the registry names may differ from the names displayed in that applet). Typical examples are “SystemAsterisk” and “NmainMouseClick”.

MIAMEX 101: Get swap wait time

```
xcall MIAMEX, MX_GETSW, waitms
```

(UNIX only) This function retrieves the current swap wait time in milliseconds. (See Function 102 below for more details).

waitms (numeric) will be returned with the current time in milliseconds, or zero to indicate that it is still set to the default (or undefined for the current terminal).

MIAMEX 102: Set swap wait time

```
xcall MIAMEX, MX_SETSW, waitms
```

(UNIX only) This function allows you to change the swap key wait time for certain terminal drivers (normally set by the SWAPWAIT parameter of the MIAME.INI, or 1000 ms by default). Currently this only affects the wyse50 driver, and was implemented because the standard wait is a full 1 second, which is apparently necessary on some versions of the terminal but not on terminal emulators. The main downside of the extra wait occurs within PolyShell, where it delayed the repaint of the new screen when swapping sessions. To allow this to remain as flexible as possible, the default value of zero simply causes the terminal driver to use its normal wait time (which in the case of wyse50 is 1000 milliseconds or 1 second). To minimize the wait, set it to 1.

waitms (numeric) should be set to the desired wait time in milliseconds, or zero to use the default.

MIAMEX 103: Get/set cmd line flags

```
xcall MIAMEX, MX_CLFLAGS, op, clflags
```

This function allows you to determine which command line switches were specified when A-Shell was launched, and to effectively set the flags that are associated with these switches.

Parameters

op (numeric)
should be set to 0 to query the current command line flags, or 1 to set them.

clflags (numeric)

will either return the current command line flags (if OP is 0) or supply the new switch settings (if OP is 1). The flags and corresponding switches are listed in the table below. (Omitted values have been reserved for internal use and are of no interest to applications.) Switches that make no sense to set at runtime (i.e. most of them) are marked "read only".

Value	Switch	Meaning
1	-v	Display version on A-Shell startup. (read only)
2	-e	Force A-Shell to exit when current command or command file is complete.
8	-d	Display A-Shell console device name on startup. (read only)
16	-k	Force prompt for new key. (read only)
256	-h	Ignore hangup signal
512	-p	Running under PolyShell (read only)
1024	-hd	Delay processing of hangup signal
2048	-hp	Send hangup signal to parent on exit
4096	-q	Quiet mode (read only)
8192	-t	Simulate slave task mode (read only)
16384	-o	Specific windows settings file specified (read only)
32768	-2	Child session (read only)
65536	-1	Disallow use of PolyShell job switching (read only)
131072	-z	Hide window (read only)
262144	-ba	Append autosnaphots to specified buffer (read only)
524288	-m	Force maximized window (read only)
1048576	-mx	Remove "X" and system menu from window (read only)
2097152	-cgi	CGI mode (read only)
83888608	-awts	ATS (A-Shell Telnet Server) mode (read only)
16777216	-hei	Generate error 250 immediately on hangup signal.
33554432	-hetcki	Treat tcki like kbd wait after hangup
67108864	-lite	AshLite mode (read only)
134217728	-min	Start minimized (read only)
268435456	-tray	Like –z but with icon in system tray (read only)

Comments

Note that when setting the command line flags, you have to set them all at once. To set just one, first retrieve the current flags and then OR in the flag you want to set, and then use *op* 1 to set the entire collection of flags.

Refer to the documentation on the A-Shell command line switches (in the *A-Shell Setup Guide*) for more information about what these switches do.

MIAMEX 104: Get process ID

```
xcall MIAMEX, MX_GETPID, pid
```

This function retrieves your current process ID number. A process ID (aka PID) is a positive integer which uniquely identifies a process. They are assigned by the operating system when the process is created, and recovered (for subsequent reuse) when the process terminates. PIDs are most useful in UNIX, where they are the most common way to identify or communicate with another process. They are less useful but still valid in the Windows environment.

pid (numeric) will return your process ID. Note that these numbers can get quite large on large systems, so you should specify at least a B,3 (or a floating point).

MIAMEX 105: Get/set clipboard text

```
xcall MIAMEX, MX_CLIPBOARD, op, text {,status {,srow, scol, erow,
      ecol}}
```

(Windows only) This function allows you to retrieve or set the text contents of the clipboard.

Parameters

op (numeric)
should be set to 0 to retrieve the clipboard text contents or 1 to set it.

text (string)
will receive the clipboard text contents for *op* 0. For *op* 1, if you want to specify a string containing the text to copy to the clipboard, then put it in the *text* parameter and do not specify the *srow*, *scol*, *erow*, and *ecol* parameters (or set them to 0). Otherwise, if you want to copy a rectangular area of the current screen to the clipboard, then specify the coordinates in the last four parameters, in which case the *text* parameter will be ignored.

status (F,6 optional)
will return 0 for success, else an error code. Currently defined codes are: -1 (unable to open clipboard – locked?); -2 (clipboard API function failed); -3 (unable to lock memory), -4 (invalid coordinates).

The clipboard provides a way for programs which do not know anything about each other to pass data. For example, you could clear the clipboard data, launch the Windows calculator (see HOSTEX), and then retrieve the clipboard contents on return, perhaps plugging it into the current data entry context.

MIAMEX 106: Get OS Information

```
xcall MIAMEX, MX_OSVER, os'name, {,osver, osrel, asplatform}
```

This function retrieves information about the current host operating system, including its name, version, and release, plus platform that this copy of A-Shell was compiled for. This can be useful in applications which wish to take advantage of features that are only available under certain platforms.

Parameters

osname (S,10+)
returns the name of the host operating system. For Windows, the possibilities are: “Win95”, “Win98”, “WinME”, “WinNT 3.51”, “WinNT 4.0”, “Win2000”, and “WinXP”. (.NET Server currently is

indistinguishable in this field from XP.) If the version information is not recognized, it will be reported as "Win X.Y" where X is the major version number and Y is the minor version number. For UNIX/Linux, OSNAME will return the same as the "uname -s" command, e.g. "Linux," "SCO_SV," "AIX."

osver (optional, S,16+)
returns the "version" of the operating system. For Windows, this might contain "OSR2" for Win95, "SE" for Win98, or "Service Pack 6" for NT. For UNIX/Linux, it will be what "uname -v" returns, e.g. a date string for Linux, or "5.0.5" (SC0), or "4" (AIX 4.3).

osrel (optional, S,16+)
returns the "release" of the operating system. For Windows, this will be the build #. For UNIX/Linux, this will be what "uname -r" returns, e.g. "2.4.2-2" (Linux), "3.2" (SCO), "3" (AIX 4.3).

asplatform (optional, S,16+)
returns the generic platform that this copy of A-Shell was compiled for. (This is the same string that would be returned in the SYSDOS field in GETJTB.SBR.) Examples are: "AIX", "SCO Unix", "Linux", "Windows/32".

MIAMEX 107: Scan user memory

Documentation update June 04; see "Update Notes" below.

```
xcall MIAMEX, MX_USRMAP, idx, name, size {,flags {,ptr}}
```

This function is analogous to the AMOS SRCH monitor call, allowing you to locate modules in user memory.

This function is used by the MAP.LIT utility, which would be the normal way to see what modules you have loaded.

Background

Although an attempt has been made to provide some of the same user memory functionality as AMOS, the design here is considerably different. The most notable difference is that under AMOS, user memory is one contiguous chunk of physical memory, and the modules are stacked end to end, starting from the lowest available address. The BASIC runtime system then typically takes over the space from the top of the last loaded module to the top of the partition. Deleting a module under AMOS causes all subsequent modules to be shifted down, which might be disastrous to a program that was currently running (and counting on the prior physical location of some memory structures).

Under A-Shell, the user memory partition (defined by the MEMORY statement in the MIAME.INI, or by the use of MEMORY.LIT or the MIAMEX function 113) is mainly used just for the BASIC runtime system's work area. Any modules, including the RUN program itself, are loaded into dynamically allocated, non-contiguous, memory chunks. Thus the nominal partition size has no real effect on the number or size of the modules which can be loaded, and loading and deleting of individual modules has no effect on any other modules or currently running programs. A-Shell keeps track of the modules which are loaded via an index table, which is analogous to a disk directory, containing the module name, size, location, and other flags.

Parameters

idx (numeric)

indicates the starting position in the memory index table to search forward from (0 to start at the beginning). It will be returned with the position of the located module, (0=not found), allowing it to be used in a subsequent call to scan forward from there.

name
should be set to the name and extension of the module to locate (e.g. "EMAILX.SBX"), or blank to locate the first module after the position specified by *idx*. It will be returned with the name of the located module.

size (numeric)
will return the size of the located module, in bytes.

flags (numeric, optional)
will return flags relating to the module status, from the table below:

Symbol	Meaning
USRMEM'INUSE	Module currently running. Cannot be deleted under any circumstances.
USRMEM'LOCKED	Module is locked in memory. This prevents it from being deleted (with a special switch) and can be set by specifying this flag on the load operation.
USRMEM'PERM	Module was manually loaded and will be left in memory until manually deleted.
USRMEM'CACHE	Module was auto-loaded and will be cached in memory for a "reasonable" amount of time.
USRMEM'UNLOCK	This is a command flag, not a module flag, but is listed here because it is needed with the delete function to delete a module that has the *usrmem'locked* flag.
USRMEM'NOFILE	Needed in order to load a memory module directly from a variable.

ptr (F,6); optional, for advanced use only!
will return the starting address of the module.

Update Notes: Build 854, 21 Nov 03

The maximum number of modules which can be loaded into user memory per job has been increased from 32 to 96.

MIAMEX 108: Load module into memory

```
xcall MIAMEX, MX_USRLOD, idx, fspec {,flags {,var}}
```

This function allows you to load a module into memory. (Note that unlike AMOS, this can be done while running a BASIC program.)

Parameters

idx (numeric)

will return the index entry number where the module was loaded, or 0 if not loaded. (-1 indicates that the module was already locked into memory.)

fspec
must be set to the specification of the file to load into memory. (If the *USRMEM'NOFILE* flag is specified, then fspec just sets the name of the module in memory.)

flags (numeric, optional)
may be set to any sensible combination of the memory module flags. Normally this would only be used to set the *USRMEM'LOCKED* flag to lock the module in memory, or to specify the *USRMEM'NOFILE* flag (which is required when loading the module from a variable).

var (unformatted, optional)
When specified in conjunction with the *USRMEM'NOFILE* flag, the module is loaded directly from the specified variable. This provides a mechanism similar to COMMON.SBR, which has the advantage of not having any particular limits on the module size or name.

See the notes under MIAMEX 107 for more information on A-Shell's user memory architecture. Also see LOAD.LIT in the *A-Shell Command Reference* which is the typical way to load a module into memory.

MIAMEX 109: Delete module from memory

```
xcall MIAMEX, MX_USRDEL, idx, name {,flags}
```

This function allows you to delete a module from user memory.

Parameters

idx (numeric)
will return the index entry number of the deleted module, if successful, or 0 if not deleted.

name
should be set to the name and extension of the module to delete (e.g. "EMAILX.SBX").

flags (numeric, optional)
may be set to *usrmem'unlock* to allow deletion of a module that is locked in memory.

See the notes under MIAMEX 107 for more information on A-Shell's user memory architecture. Also see DEL.LIT in the *A-Shell Command Reference*, which is the typical way to load a module into memory.

MIAMEX 110: Save memory module

```
xcall MIAMEX, MX_USRSAV, status, name
```

This function allows you to save a memory module to a disk file.

Parameters

status (F,6)

will return the number of bytes written to disk if successful, 0 if the module was not found in memory, and a negative number to indicate an error. Currently defined error conditions are: -1 if there was a discrepancy in the number of bytes written (perhaps indicating a disk quota or limit was exceeded); -2 if the file could not be opened (perhaps indicating a privilege or write protect problem); and –3 if there was an error in processing the file specification (perhaps indicating that you are not logged into a valid directory).

name
should be set to the name and extension of the module to save (e.g. "EMAILX.SBX"). It will be saved to your current logged in directory.

See the notes under MIAMEX 107 for more information on A-Shell's user memory architecture. Also see SAVE.LIT in the *A-Shell Command Reference*, which is the typical way to load a module into memory.

MIAMEX 111: Read/write memory module

```
xcall MIAMEX, MX_USRIO, status, module, op, var, offset ,bytes
```

This function allows you to read and write directly to a memory module. This is a relatively advanced technique that might be useful either to maximize performance in an I/O intensive operation, or to gain the equivalent functionality of a dynamically allocated unformatted variable. (For the latter, it would be necessary to first create a dummy disk file of the desired size, then use MIAMEX function 108 to load it into memory.)

Parameters

status (F,6)
will return the number of bytes successfully transferred (0 if the module was not found; negative numbers for errors).

module
must be set to the name or index number of the module to access. If it is a string, it is interpreted as the name and extension of the module (e.g. "EMAILX.SBX"). If it is a numeric parameter, it is interpreted as the index number of the module. (The index number can be obtained from the module name and extension using MIAMEX function 107, and its subsequent use would be more efficient by eliminating the need to scan the memory module list to locate the module by name on each access.)

op (numeric)
indicates the operation: 0 for read and 1 for write. 2 and 3 are equivalent to 0 and 1, respectively, except that do automatic unblocking of records, exactly the way disk file records are unblocked when the record size is less than 512 bytes and the SPAN'BLOCKS option is not used. The record size is assumed to be the size of the *var* parameter, unless the *bytes* parameter is specified.

var (unformatted)
provides the data to be written or receives the data read.

offset (numeric, optional)
specifies the starting offset from the beginning of the memory module. If *op* is 0 or 1, the offset is assumed to be in bytes. If *op* is 2 or 3, it is in records (of size defined by the size of the *var* parameter, or the value of the *bytes* parameter, if specified).

bytes (numeric, optional)
indicates the number of bytes to read or write. In record mode (*op* codes 2 and 3) it may be omitted, in which case the size of the *var* parameter will determine this.

See the notes under MIAMEX 107 for more information on A-Shell's user memory architecture.

MIAMEX 112: Set AutoMouse translations

```
xcall MIAMEX, MX_AMOUSEXLT, op, status, flags, s1, t1, ... s10, t10
```

This function allows you to define or otherwise control AutoMouse translations within a program. AutoMouse is an A-Shell scheme, which can be activated or deactivated from the Edit menu, which allows the user to double-click on a text "token" anywhere on the screen to transmit those characters to the keyboard. A "token" is a string of characters delimited by any non-alpha, non-numeric characters (i.e. space or punctuation). This MIAMEX function allows you to extend or control this capability by creating translation strings that are not limited to "tokens" (as just defined) and which can transmit arbitrary text (rather than the characters of the string which is double-clicked on).

Parameters

op (numeric)
should be set to one of the following:

Value	Operation
0	Add (or create new) translation list
1	Clear existing translations
2	Temporarily disable translations
3	Re-enable translations

status (F,6)
returns 0 on success, else an error code. Currently defined errors are –1 (cannot allocate memory for translations) and –2 (translations exceed memory limit of 2048 bytes).

The remaining arguments are only used if OP=0.

flags (numeric)
affect the way the translations work. Currently the only defined flag value is 1, which indicates that the translation is not case sensitive.

s1...s10
strings representing the text which may appear on the screen, to be clicked on and translated (to the corresponding *t1...t10* values). Note that these strings may contain spaces and other non-alphanumeric characters, which would otherwise be considered delimiters. The maximum length of each string is 132 characters.

t1 thru *t10*
strings containing the characters to be transmitted to the keyboard when the corresponding string (in *s1* thru *s10*) is double-clicked. They may also be specified as B or F variables, in which case the

define the byte value of a single byte to be transmitted. (This may be more convenient than string representation when dealing with control characters, e.g. 1=Control-A, 27=Escape, etc.).

You do not have to specify all 10 pairs of *sn* and *tn* strings, and even if you do, interpretation will stop at the first null *sn* string. There is no particular limit to the number of translation pairs you can define (using as many calls to this function as necessary). The total combined limit on the *sn* and *tn* strings is 2048 characters.

MIAMEX 113: Change memory partition

```
xcall MIAMEX, MX_MEMORY, newsizk, status
```

This function allows you to change the size of your user memory partition on the fly.

Parameters

newsizk
is the desired new size (in Kilobyte units, e.g. 1024 = 1024K).

status (F,6)
will be set to 0 if successful.

MIAMEX 115: Indirect XCALL

```
xcall MIAMEX, MX_IXCALL, sbrnam, xcbadr, status
```

This function allows you to forward the calling parameter from one BASIC **Error! Reference source not found.** subroutine to another, evaluating the new subroutine name on the fly.

Parameters

sbrnam (string)
is the name of the new subroutine to call.

xcbadr (floating point)
is the address of the parameter structure which was passed to the original routine. (See the documentation on writing your own subroutines for more information on the XCALL parameter structure.)

status (F,6)
will return 0 for success, >0 for subroutine not found or unable to load, else <0 for other system errors.

Comments

This routine is used in the RXCALL.**Error! Reference source not found.** routine, which allows you to remotely call a routine running on another machine. See the documentation on writing your own subroutines for more information about RXCALL.

MIAMEX 116: Process color ini file

```
xcall MIAMEX, MX_INICLR, fspec, status
```

This function allows you to process (or reprocess) the specified color initialization file. Normally, this file is called DSK0:INI.CLR[7,0] and is processed automatically when A-Shell is launched. But it may be useful to be able to edit this file and reprocess it, or switch to a different color settings file, without restarting A-Shell.

fspec must contain the file specification (AMOS or native format) of the color settings file to process. (If blank, the default is DSK0:INI.CLR[7,0]. If no extension is specified, the default extension is CLR.

Refer to the *A-Shell Setup Guide (Configuration)* for details on the format of the INI.CLR file.

MIAMEX 117: Send MAPI mail

```
xcall MIAMEX, MX_MAPI, status, fspec {,flags, subject, text}
```

(Windows only) This function provides a simple way to send the contents of a file via interactive email. It launches the current email client program (e.g. Outlook, Eudora, etc.) and then inserts the contents of the file into the body of the message. The user has only to address the message and optionally type in introductory comments.

Parameters

status (F,6)

will return a status code indicating the success or failure of the operation, from the table below. Note that the negative error numbers relate to errors within the MIAMEX subroutine, while the positive errors are returned from the MAPI interface itself.

Value	Meaning		Value	Status
0	Success		11	MAPI Attachment not found
-1	MAPI interface not available on this computer		12	MAPI Error opening attachment
-2	Unable to load MAPI32.DLL		13	MAPI Error writing attachment
-3	Unable to get address of MAPISendMail function		14	MAPI Unknown recipient
-4	Unable to allocate memory		15	MAPI Bad receipt type
-5	Unable to open *fspec*		16	MAPI No messages
-6	*fspec* not found		17	MAPI Invalid message
1	MAPI User abort		18	MAPI Text too large
2	MAPI General failure		19	MAPI Invalid session
3	MAPI Logon failure		20	MAPI Type not supported
4	MAPI Disk full		21	MAPI Ambiguous recipient
5	MAPI Insufficient memory		22	MAPI Message in use
6	MAPI Access denied		23	MAPI Network failure
8	MAPI Too many sessions		24	MAPI Invalid edit fields

Value	Meaning		Value	Status
9	MAPI Too many files		25	MAPI Invalid recipients
10	MAPI Too many recipients		26	MAPI Not supported

fspec
is the file specification (AMOS or native format) of the file to send via email.

flags (optional, numeric)
may contain any combination of the following values:

Value	Meaning
1	Receipt requested
4	Send using HTML fixed pitch
12	Very small fixed type

subject (optional, string)
may be set to the desired subject of the message. (Otherwise the user can type a subject in interactively.)

text (optional, string)
may be set to any text which will be inserted in the message prior to the specified file. (This might be useful for introductory comments.)

To send email without user interaction, an A-Shell add-on module call EMAILX.SBX is available from MicroSabio.

MIAMEX 118: Get/set stream position

```
xcall MIAMEX, MX_FILEPOS, chan, op, pos {,filidx}
```

This function allows you to get or set the current file position within a sequential file which is open for input or output.

Parameters

chan (numeric)
should be set to the channel number of the open sequential file.

op (numeric)
must be set to 0 to get the current file position or 1 to set it.

pos (F,6)
is the position (offset in bytes from the start of file). For OP 0 (get), the current position is returned in this parameter. For *op* 1 (set), *pos* must be set to the desired new file position.

filidx (B,4)
is an optional handle to a "file index" which can be created and passed back to you by XTREE. If *filidx* is specified and *op* = 1, then rather than setting the file pointer to the byte position specified by

pos, it instead interprets pos as the desired line number, and uses the index referenced by *filidx* to locate that line number directly (without having to scan the file sequentially).

If there is an error in the operation, *pos* will be returned as -1.

MIAMEX 119: Manage controls

This is identical to the AUI Control class, which should be used instead and which see for further information.

MIAMEX 120: Prompt for Windows printer

```
xcall MIAMEX, MX_WINPTR, status, printer {,port {,driver}}
```

(Windows only) This function displays a standard Windows printer selection dialog and returns information about the printer selected. This might be useful for building printer initialization files (see Printer Configuration in the *A-Shell Setup Guide*) based on the selected printer. Or, as of A-Shell/Windows build 827, you can take it one step farther and just use the full Windows printer name (as returned in the *printer* parameter) directly when sending files to the printer. See EZSPL in this document, and PRINT.LIT in the A-Shell Command Reference, for more information.

Parameters

printer (string)
: will return the descriptive printer name of the printer selected by the user. (This is normally the name which appears under the icon in the printer selection window.)

port (optional, string)
: will return the port name. This could be a traditional physical port (like LPT1:), or a logical network port (like NET01: or IP_192.158.200.250).

driver (optional, string)
: will return the name of the driver. In most cases with newer versions of Windows, this will be the universal spooler driver "winspool" (which in turn calls the hardware-specific driver.)

MIAMEX 121: Set/get chain-to on priv error

```
xcall MIAMEX, MX_CHAINTO, op, progrm
```

This function allows you to define a program to chain to automatically if an attempt is made to run a program for which the user does not have sufficient (read) privileges. This is useful if you wish to use the operating system file access privileges for restricting access to sensitive programs (as an alternative, for example, to password protecting them). If no chain-to specification is defined, then such an attempt will cause the program to abort, displaying an "access denied" message, probably leaving the user at the dot prompt. By defining a chain-to program, when such an error occurs, the user will automatically be redirected to the specified program (which might be a menu, or perhaps an error recovery program).

Parameters

op (numeric)
should be set to 0 to retrieve the current chain-to specification into the *progrm* variable, or 1 to set the new chain-to specification from the contents of the *progrm* argument.

progrm (string)
will either return (for OP 0) or be used to set (for OP 1) the chain-to specification. It should be a RUN or LIT program in AMOS format (no CMD or DO files allowed). Examples would be "`MAIN.RUN`" or "`DSK0:MAIN.RUN[100,22]`" or "SYS:HOST.LIT."

Comments

If the application allows the user to move around to different PPNs, it would be wise to include the complete specification for the desired chain-to program (else it might not be found). Also note that even though the program can be found, the user may no longer be in the expected directory, so a robust program would probably want to log the user to the expected directory automatically (see LOG).

MIAMEX 124: Output to ashlog.log

MIAMEX 124 was added to A-Shell in build 862, 02 Feb 04.

```
xcall MIAMEX, MX_ASHLOG, string
```

This function outputs the *string* message to the ashlog.log file. It is useful when you want to log your own messages in the same stream that A-Shell uses for its system messages.

MIAMEX 125: Get last MCRS click info

MIAMEX 125 was added to A-Shell in build 862, 02 Feb 04.

```
xcall MIAMEX, MX_MCRS, clickinfo, row, col
```

This function retrieves information about the last mouse click (when mouse cursor reporting has been activated via TCRT 158). *clickinfo* returns a bitmap consisting of the following:

```
1   Left button
2   Right button
8   Double click (else single click)
```

For example, 1 is a standard left click, while 10 is a right doubleclick. *row* and *col* indicate the row,col position of the click.

MIAMEX 126: Sink/unsink field

MIAMEX 126 was added to A-Shell in build 866, 12 Feb 04.

```
xcall MIAMEX, MX_SINK, op, srow, scol, erow, ecol, bgc
```

Causes the specified box to be “sunk” or “unsunk”, based on OP (1 to sink, 0 to unsink). This is the same routine used by INFLD to give the field being editing the appearance of being slightly sunken. You can also get this effect by using the MBF'SUNKEN flag with MIAMEX 119 to create a static text control with that style, but in that case, you can't freely write on top of the control. With MIAMEX 126, the effect does not conflict with character-level I/O to that screen area (which is why it works with INFLD).

MIAMEX 127: Get/set rounding factor

MIAMEX 127 was added to A-Shell in build 878, 02 April 2004.

```
xcall MIAMEX, MX_ROUND, op, factor
```

This function allows you to get (*op* = 0) or set (*op* = 1) the new variable rounding factor. Basically the factor is set to the level of precision you would like. 1 means round all variables to the nearest integer. .01 would be to the nearest one hundredth. 0 disables the feature. Note that the setting persists across programs. For an in-depth discussion of the floating point rounding issue, see the topic entitled *Rounding of Floating Point Variables* in the *A-Shell Development Guide*.

MIAMEX 128: Get IP address

MIAMEX 128 was added to A-Shell in build 879, 12 April 2004.

```
xcall MIAMEX, MX_GETIP, ip$
```

Returns the IP address of this computer.

MIAMEX 129: GUI Environment

This is identical to the AUI Environment class, which should be used instead and which see for further information.

MIAMEX 130: Get startup command

MIAMEX 130 was added to A-Shell in build 884, 01 May 2004.

```
xcall MIAMEX, MX_ASHFILE, startcmd$ {,ashfile$}
```

(Windows only) This function returns the startup command (i.e. the initial command that is executed on startup when A-Shell gets to the dot prompt) into STARTCMD$. If the ASHFILE$ argument is specified, it returns the name of the settings (.ash) file that was specified on the ashw32 command line (-o parameter).

MIAMEX 131: Get file stats

MIAMEX 131 was added to A-Shell in build 885, 03 May 2004.

```
xcall MIAMEX, MX_FILESTATS, loc'rem, path, bytes, mtime, ctime, mode
```

This function returns stats for the specified PATH, either on the local system or the remote PC via ATE. Parameters are as follows:

Parameters

loc'rem (S,1)
"L" for local system, "R" for remote PC

path (S)
specifies a native path or AMOS-style filespec. If native, may include embedded environment variables using the %env% syntax. Note that if LOC'REM is "R," and AMOS specification in PATH will be interpreted relative to the INI file used by ATE on the PC.

bytes (F)
returns the size of the file. If PATH is not found BYTES will be set to -1. -2 indicates that you are attempting to use a remote PATH when you don't have ATE support.

mtime, ctime (F or B,4)
return the last modification time and creation time, in seconds since "the epoch" (Midnight Jan 1, 1970). This format is very convenient for comparing filetimes, but be forewarned that comparing filetimes on different machines (e.g. to decide if a file transfer/update is needed) requires that you synchronize the clocks on the machines in question! If you want to display the filetimes in a more human-friendly format, use MIAMEX function 132.

mode (F or B,2+)
returns the mode bits indicating the type and other attributes of the file. The most interesting bits are:

Value	Meaning
1	execute privilege (other)
2	write privilege (other)
4	read privilege (other)
8	execute (group)
16	write (group)
32	read (group)
64	execute (owner)
128	write (owner)
256	read (owner)
4096	pipe
8192	chr/special
16384	directory
32768	regular file

MIAMEX 132: Format file time

MIAMEX 132 was added to A-Shell in build 885, 03 May 2004.

```
xcall MIAMEX, MX_FTFORMAT, ftime, string
```

This function is used to format a numeric filetime of the type returned by MIAMEX,131 into a more friendly format.

MIAMEX 133: Expand file in place

MIAMEX 133 was added to A-Shell in build 893, 24 June 2004.

```
xcall MIAMEX, MX_EXPFIL, ch, addrecs, status
```

Parameters

ch
: is the channel number that the file is open (for random exclusive access on). If running LOKSER, you should probably open the file for RANDOM (exclusive) before expanding it. If not running LOKSER, you are free to use some other locking scheme. Although the XCALL will not stop you from expanding a file without first gaining exclusive access, you are likely to run into synchronization conflicts with other users who have the file open. For one thing, those users will not be able to access the expanded area (until they close and reopen the file), so if you update a control record they may try to access the new area and get an illegal record number.

addrecs
: is the number of logical records to add (based on the record size passed in the file open statement.) It will be converted to the appropriate number of 512 byte blocks to add. (The added blocks will be filled with]]] characters.)

status
: returns the new total number of logical records in the file if the operation is successful. 0 indicates a parameter or other configuration error trapped by the XCALL (generally accompanied by a displayed message.) <0 indicates an operating system error #, which you can retrieve the message for using MIAMEX,86,STATUS,MSG

Note that it is up to you to update any control records or other internal structures to correspond to the new file size. Remember to release your exclusive access after doing so.

MIAMEX 135: Eventwait

This is identical to the AUI Eventwait class, which should be used instead, and which see for further information.

MIAMEX 137: Windows help system

This is identical to the AUI HTMLHelp class, which should be used instead, and which see for further information.

MIAMEX 138: Registry operations

MIAMEX 138 was added to A-Shell in build 908.2, 17 November 2004.

```
xcall MIAMEX, MX_REGISTRY, opcode, <params depending on opcode>
```

This function supports a range of Registry operations as noted in the table below.

Opcode	Function
REGOP_OPEN	Open
REGOP_CREATE (2)	Create
REGOP_SET (3)	Set
REGOP_READ (4)	Read
REGOP_ENUMKEYS (5)	Enumerate keys
REGOP_ENUMVALS (6)	Enumerate values
REGOP_CLOSE (7)	Close
REGOP_DELIMS (64)	Replace Null Delimiters

Update Notes: Build 925 - 30 March 2005

In this release, XCALL MIAMEX,138 was enhanced to provide an option for more friendly delimiters when retrieving or setting REG_MULTI_SZ values. This value type is used for lists of names, like bin names or paper size names, and consists of strings separated by a null byte, with the last string terminated by two null bytes. This format makes sense in C, but in Basic, it is difficult to deal with the embedded nulls. The new option replaces the null terminators, except for the very last one, with chr(128) characters. For the set opcode, you may optionally use these same chr(128) characters to build your value string. The option is activated by adding 64 (REGOP_DELIMS) to the opcode. This applies to opcodes 4 and 6.

See the sample program REGPTR.BP for a working example.

Open

This call is needed to open up a branch of the Registry and returns a handle which can be used in subsequent opcodes.

```
xcall MIAMEX, MX_REGISTRY, REGOP_OPEN, hkey, subkey, rights,
      hkeynew, status
```

Parameters

hkey (b,4)

can be a key returned in the *hkeynew* parameter from a previous open, or more likely, one of the following symbols (from **Error! Reference source not found.**) for the main hives in the Registry:

Symbol
HKEY_CLASSES_ROOT
HKEY_CURRENT_USER
HKEY_LOCAL_MACHINE
KEY_USERS

subkey
is a string combining one or more levels beneath the level referenced by the key *hkey*, for example:

```
SUBKEY = "SOFTWARE\MicroSabio\JBCT\ATE"
```

Note that *subkey* is not case sensitive on lookup, but when creating keys, the case specified will be retained. (Same idea as with Windows filenames.)

rights
is a numeric variable combining one or more of the following bits relating to the kind of rights you want to request for this key:

Symbol	Description
RGKEY_QUERY_VALUE	Ability to query a value
RGKEY_SET_VALUE	Ability to change or set a value
RGKEY_CREATE_SUB_KEY	Ability to create a new sub key
RGKEY_ENUM_SUB_KEYS	Ability to enumerate sub keys

hkeynew (b,4)
will return the handle to the opened key, and will be needed for subsequent operations.

status (f,6)
returns 0 for success, or else an error code. You can convert the error code to a message using:

```
xcall MIAMEX,MX_ERRNOMSG,status,msg
```

Create

This is used to when saving values to the Registry. It will create the specified subkey if it doesn't exist, or if the subkey already exists, it will open it.

```
xcall MIAMEX, MX_REGISTRY, REGOP_CREATE, hkey, subkey, rights,
      hkeynew, status
```

The parameters are nearly the same as for Open. The main difference is that the *status* parameter can return either of the following:

```
STATUS=1 (Key was created)
STATUS=2 (Key already existed and was opened)
```

Set

This is used to write a new or updated value to the Registry.

```
xcall MIAMEX, MX_REGISTRY, REGOP_SET, hkey, name, type, value,
      status
```

Parameters

hkey (b,4)
must be set to the *hkeynew* value returned from a previous call to Create or Open a key.

name (string)
is the name of the value to write. (These are the items that appear in the left side of the right hand pane of REGEDIT.EXE.)

type (numeric)
specifies the type of value you want to write:

Types	Value	Description
REG'NONE	0	No type
REG'SZ	1	Null terminated string
REG'EXPAND'SZ	2	Same as (1) but signals that string may contain embedded environment vars (e.g. "%TEMP%\xyx.log")
REG'BINARY	3	Raw binary data (use X format)
REG'DWORD	4	4 byte integer (use B,4 format)
REG'MULTI'SZ	7	List of null terminated strings, with a double null terminating the list. See Replace Null Delimiters for an alternative to using null delimiters (with are hard to work with in Basic).

value (string, unformatted, or B,4 depending on *type*)
The data to write to the named item.

status (f,6)
0 for success, else error code.

Read

Read is used to read a single value from the Registry.

```
xcall MIAMEX, MX_REGISTRY, REGOP_READ, hkey, name, type, value,
      status
```

Parameters

hkey, name
must be set as in opcode 3 (Set)

type

is ignored on input, and is updated by the call to indicate the data type of the value returned (see table of data types under Set).

value (string, unformatted, or b,4 depending on the type of the data)
returns the data. Note that the application is expected to either know the type of data, in which case it can supply an appropriate form of *value*, or it can use the overlay technique and then extract the data based on the returned *type* field. For example, you might map *value* as:

```
MAP1 VALUEX,X,512
MAP1 VALUE$,S,512,@VALUEX
MAP2 VALUE,B,4,@VALUEX
```

status (f,6)
0 for success, else error code.

Enumerate keys

This is used to list the subkeys of the opened key.

```
xcall MIAMEX, MX_REGISTRY, REGOP_ENUMKEYS, hkey, subkey, index,
      status
```

Parameters

hkey
must be set as in opcode 3 (Set)

subkey (string)
will return the subkey name.

index (numeric)
should be set to 0 for the first subkey, and incremented for each subsequent subkey. There is no particular order to the subkeys, so do not put any special significance on *index*.

status (f,6)
0 for success, else error code. Error 259 indicates that there are no more keys to enumerate.

Enumerate values

This is used to list the named values for the opened key.

```
xcall MIAMEX, MX_REGISTRY, REGOP_ENUMVALS, hkey, name, index, type,
      value, status
```

Parameters

hkey
must be set as in opcode 3 (Set)

index (numeric)

should be set to 0 for the first subkey, and incremented for each subsequent subkey. There is no particular order to the subkeys, so do not put any special significance on *index*.

name (string)
will return the value name corresponding to *index*.

type (numeric)
will return the type of the value (see types under Set).

value (type depends on *type*)
will return the value of the named item. See Read for notes on how to map it to support multiple types.

status (f,6)
0 for success, else error code. Error 259 indicates that there are no more values to enumerate.

Close

This function is used to close a handle previously opened.

```
xcall MIAMEX, MX_REGISTRY, REGOP_CLOSE, hkey, status
```

hkey
must be set to the handle of the key to close (as returned in HKEY2 by the open or create operations).

Replace Null Delimiters

This is not a function but an option which can be added to the Set, Read, and Enumerate values functions to simplify the handling of the REG_MULTI_SZ data type. Without this option, in order to parse a list of strings containing embedded null bytes, you need to overlay the string variable on an unformatted variable and then use substring referencing to locate the null bytes and access the substrings between the null bytes. When the REGOP_DELIMS flag is added to, say, REGOP_ENUMVALS, then the embedded nulls are replaced by CHR(128) characters. Such a list of strings can then be parsed with code similar to the following (excerpted from the sample program REGPTR) which lists the individual bin names available for a printer.

```
! enumerate next value . . .
xcall MIAMEX, MX_REGISTRY, REGOP_ENUMVALS + REGOP_DELIMS, HKEYNEW, SUBKEY, INDEX,
   TYPE, V$, STATUS

! if no error, and value is printBinNames and of REG_MULTI_SZ type, parse it out
If STATUS=0 and SUBKEY="printBinNames" and TYPE=REG_MULTI_SZ then
   X = 1
   Y = 1
   do while Y > 0
      Y = instr(X,V$,chr(128))
      if Y > 0 THEN
         BIN$ = V$[X,Y-1]
         X = Y + 1
         Print "BIN Name = ";BIN$
      endif
   Loop
endif
```

MIAMEX 141: Set Parent Control

MIAMEX 141 was added to A-Shell in build 909.7, 19 December 2004.

```
xcall MIAMEX, MX_AUTOPARENT, ctrlid
```

This function allows you to set the default parent control, to apply to subsequent TPRINT/DPRINT and TAB(-1,9) and TAB(-1,10) commands.

Although you can accomplish the same result using the AUI Control functions and specifying the parent control explicitly, the MX_AUTOPARENT function greatly simplifies dialog coding by allowing you to use the much simpler TPRINT, DPRINT, and TAB(-1,9-10) commands. This is particularly true if SET AUTOTPRINT is used to interpret existing PRINT statements as if they were TPRINT statements.

MX_AUTOPARENT can be used to set the parent to a group box (rather than a dialog), but in this case, only the TPRINT and DPRINT commands are affected.

You should clear the auto parent setting when no longer needed, by calling the routine with ctrlid set to 0. It will also be cleared automatically at the start of a new RUN program.

MIAMEX 146: Get Last Line Number

MIAMEX 146 was added to A-Shell in build 931, 19 May 2005.

```
xcall MIAMEX, MX_LASTNO, lineno
```

This function allows you to find out the last line number executed in the current program. This is similar to the err(1) function in an error trap except it doesn't require that an error occur.

MIAMEX 147: Count Lines in File

MIAMEX 147 was added to A-Shell in build 931, 22 May 2005.

```
xcall MIAMEX, MX_FLINES, channel, lines {,linelen {slinelen}}
```

This function allows you to quickly count the number of lines in a sequential file, and also determine the length of the longest line either with or without trailing spaces.

Parameters

channel (in, numeric)
: must be set to the file channel number of a file open for sequential input.

lines (out, numeric)
: returns the number of lines in the file. A negative number indicates an error (-1 means the file channel was invalid, -x indicates that system error x occurred while reading the file.)

linelen (out, numeric, optional)
: returns the length of the longest line in the file (including any line terminators such as CR and/or LF).

slinelen (out, numeric, optional)
returns the length of the longest line in the file excluding any trailing spaces

The file is left open, with the file cursor set to the beginning of the file.

MIAMEX 148: Calculate String Length or Height

MIAMEX 148 was added to A-Shell in build 932, 31 May 2005.

```
xcall MIAMEX, MX_GDICALC, opcode, handle, status {,params}
```

(Windows/ATE) This function (148 or MX_GDICALC) performs various calculations on the length and/or height of a string as it will appear when printed. In the case of a rectangle, it will also break the string into two parts, the first of which will fit within the rectangle, the remainder of which will be printed in another space or on the next page.

The *handle* variable (F6 or B4) is returned by the open operation (OP 1) and must be specified to all the others. *status* will be returned from each operation, with 0=success; all others are errors, with the most typical being:

Error	Description
-1	Unable to open printer context (bad printer name or failure to first open printer)
-2	Bad map mode
-3	Error setting font
-4	Insufficient number of parameters passed
-5	Illegal opcode
others	Negative of the system error number

Opcode	Function
MXGDI_OPENPTR	Open a printer device context
MXGDI_SETFONT	Choose font
MXGDI_CALCRECT	Calculate rectangular text metrics
MXGDI_CALCLEN	Calculate string length
MXGDI_CLOSEPTR	Close printer context. This must be done BEFORE printing to it.

See the sample program MMOGDI.BP and the updated sample text file TSTGDI.TXT in the SOSLIB for examples of using XCALL MIAMEX,MX_GDICALC and /TEXTRECTANGLE.

Open

```
xcall MIAMEX, MX_GDICALC, mxgdi_openptr, handle, status, printer,
     mapmode
```

PRINTER (string) may be any form of printer name allowed by SPOOL.SBR (e.g. a null string for the A-Shell or Windows default; a printer init file name, a printer share name, or a printer local driver name). MAPMODE must be one of the map modes allowed in the GDI //SETMAPMODE statement (LOENGLISH, HIENGLISH, LOMETRIC, HIMETRIC, TEXT, TWIPS, DECIPOINTS).

Choose font

```
xcall MIAMEX, MX_GDICALC, mxgdi_setfont, handle, status, pointsz,
      {,face, pitchfam, charset, wt, style}
```

This is equivalent to the GDI //SETFONT statement (which see in the *A-Shell Development Guide* for parameter information). Once the font is set, it remains set until changed with another OP 2 call.

Calculate rectangular text metrics

```
xcall MIAMEX, MX_GDICALC, mxgdi_calcrect, handle, status,  lft, top,
      rght, btm, memo$, height, overflow$
```

OP 3 calculates how much of the text in the MEMO$ variable will fit inside the rectangle defined by LFT,TOP,RGHT,BTM. If it all fits, then HEIGHT will be returned set to the height needed. Otherwise, MEMO$ will be truncated at the length that will fit, and the remainder will be returned in OVERFLOW$. HEIGHT will be set to BTM-TOP (i.e. the height of the specified rectangular area.) This command is useful in conjunction with //TEXTRECTANGLE for determining in advance how tall the rectangle must be (either for drawing a border rectangle with //RECTANGLE, or for breaking a long memo in half, putting the remainder on another page.)

Calculate string length

```
xcall MIAMEX, MX_GDICALC, mxgdi_calclen, handle, status,  memo$,
      length
```

OP 4 is similar to OP 3, but just tells you how long the specified string will be when printed in the current printer/font. MEMO$ is assumed to be a simple string with no embedded CRLF characters.

```
xcall MIAMEX, MX_GDICALC, mxgdi_closeptr, handle, status
```

MSBOXX

Documentation update June 04; see "Update Notes" below.

```
xcall MSBOXX, strow, stcol, endrow, endcol, boxcod {, boxsts,
      boxclr}
```

MSBOXX.SBR is nearly equivalent to the version under AMOS (developed by MicroSabio as part of the TRACKER package), except that it does not support the *boxmap* and *boxctl* parameters of the AMOS version. For complete details, you should consult the *TRACKER User's Guide*. Here, however, is a brief summary:

Parameters

strow, stcol, endrow, endcol

specify the coordinates of the upper left and lower right corner of the box to be drawn. It is important to note that if a border is applicable, it is drawn OUTSIDE of the coordinates specified for the box. (Thus the interior, or usable part of the box is the same size whether or not the border option is specified.)

boxcod

specifies one or more options (added together). These are generally referenced symbolically via the MSBOXX.BSI include file.

Value	Symbol	Meaning
1	BOX'ERA	Clear interior of the box
2	BOX'BDR	Draw border around the box
4	BOX'SVA	Save area used by box (to be restored later)
8	BOX'RSA	Restore area (previously saved)
16	BOX'COF	Leave cursor off on exit
32	BOX'REV	(Not supported under A-Shell)
64	BOX'FAO	Field attributes on/off (at edge of box)
128	BOX'CHK	Return BOXSTS=0 if save/restore supported
256	BOX'PSA	Pop saved area without displaying it
1024	BOX'MAP	(Not supported under A-Shell)
2048	BOX'PRT	(Not supported under A-Shell)
4096	BOX'ATR	Save/restore screen context
8192	BOX'HLI	Draw horizontal line (set *strow* = *endrow*)
16384	BOX'VLI	Draw vertical line (set *stcol* = *endcol*)
32768	BOX'DBL	Draw double line border around box
65536	BOX'SBU	Scroll box up one line
131072	BOX'SBD	Scroll box down one line

boxsts

optionally returns a code indicating if the operation succeeded. 0 indicates success.

boxclr

optionally defines the set of colors to use for the parts of the box:

```
MAP1 BOXCLR
   MAP2 BRDR'FG,B,1    ! Border foreground
   MAP2 BRDR'BG,B,1    ! Border background
   MAP2 IBOX'FG,B,1    ! Interior foreground
   MAP2 IBOX'BG,B,1    ! Interior background
```

Comments

Since TRACKER is built-in to A-Shell, you can always count on the ability to save and restore screen areas with **MSBOXX.SBR**. You can also perform all of these box drawing and save/restore operations using individual TAB(x,y) commands, but with considerably more effort.

Update Notes: Build 863.4 - 02 Feb 04

(Windows) A new opcode option has been added to **MSBOXX.SBR** to display a GUI-style pop-up window: 524288. This has been added to MSBOXX.BSI as BOX'WIN. When specified, and using the GUI version of the driver, it causes the pop-up box to be created using a static text control with the MBF'FRAME option and the standard "button face" background color (generally gray). The "frame" is much thinner than the normal MSBOXX border, allowing the rows and columns which are normally taken by the border to be used for text if so desired. This style of box is the only kind that can overlap other controls (without causing the underlying controls to have to be hidden). However, if you use this style of box, you should only print within it using TPRINT, DPRINT, or MIAMEX 119. Any other kind of output will "print through" to the underlying screen layer and not be removed when the box is removed.

Update Notes: Build 884.0 - 28 Apr 04

The operation of the MSBOXX opcode BOX'WIN (to create a GUI-style raised panel box) has been improved to no longer require that the border option be selected, and more importantly, to work over ATE. If selected in a non-GUI environment, the option is just ignored (so you should combine it with whatever other box options you would otherwise use, including BOX'SVA, BOX'RSA, BOX'ERA, etc.)

Note that one of the advantages of BOX'WIN for pop-up boxes is that being a true control, it can overlay other controls, and other controls can overlay it, without the complications that arise when trying to save and restore text boxes in a mixed GUI environment. However, if you do use BOX'WIN, you should only use GUI-style static text and other controls to write within the box.

MSGBOX

Documentation update June 04; see "Update Notes" below.

```
xcall MSGBOX, msg, title, btnflag, iconflag, miscflgs, rtncde
```

MSGBOX. Error! Reference source not found. displays a message in a dialog box format, with a variety of standard response button options. Under Windows, these messages use the standard Windows message box facility, which uses a pop-up floating window. Under UNIX, they are implemented via the INMEMO free-form menu mode, which is also pop-up but cannot be dragged with the mouse.

Parameters

msg (string, up to 1024 bytes).

Specifies the message to display. The width of the message box will be adjusted (within certain reasonable limits) to the size of the message, which will them be wrapped as needed. You can also force your own line breaks and blank lines by inserting CHR$(13) characters.

title (string, up to 128 bytes)

Specifies the title that will appear in the title bar of the message box.

btnflag (numeric)

Select one of the following values to specify the set of response buttons that will appear on the message box:

Value	Button Flag Description
0	OK button only
1	CANCEL button only
2	ABORT, RETRY, and IGNORE buttons
3	YES, NO, and CANCEL buttons
4	YES and NO buttons
5	RETRY and CANCEL buttons

iconflag (numeric)

Select one of the following to specify the icon that appears in the message box. (Ignored under UNIX).

Value	Icon Flag Description
16	STOP (Red X)
32	QUESTION (?)
48	EXCLAMATION (!)
64	INFO (i)

mscflags (numeric)

Combine zero or more of the following to specify misc options:

Value	Misc Flags Description
256	Make the second button be the default. (Normally the first button will be the default button.)
512	Make the third button be the default.
4096	Task modal. Suspends the A-Shell session until the dialog box is responded to. (Windows only)
8192	System modal. Suspends all Windows applications until the dialog box is responded to. (Windows only)

rtncde (numeric)

Returns a code indicating what button was pushed to respond to the message, from the list below:

Value	RTNCDE meaning
1	OK button
2	CANCEL button
3	ABORT button
4	RETRY button
5	YES button
6	NO button
7	Message box was closed without clicking on one of the button choices (either the X button in the corner of the dialog under Windows, or the ESC key under UNIX)

Comments

All of the parameters, including all of the flag values, are mapped in the ++include file MSGBOX.MAP, which is also included with the standard release, and which should be included in programs using MSGBOX.SBX to make the coding more intelligible.

Update Notes: Build 850.1 - 04 Oct 03

MSGBOX.SBX 1.0(101) now supports a HELP button in message boxes. To add a HELP button, add 16384 to the *btnflag* argument to MSGBOX.SBX. The subroutine returns 9 if the button is clicked.

MSGLOG

```
xcall MSGLOG, text, code, status
```

MSGLOG.SBR is equivalent to the version under AMOS, except for the format of the OPR:SYSLOG.SYS file. Under AMOS, the file is binary coded and requires **SYSLOG.LIT** to format and display it. Under A-Shell, the file is a simple text file which can be printed or displayed directly.

NFIND

```
xcall NFIND, string, stpos, endpos, char, pos
```

NFIND.SBR is similar to the BASIC INSTR() function, except with more sophistication.

Parameters

string
is the string to be searched. *stpos* (any numeric type) is the starting position (first byte is #1) and *endpos* (any numeric type) the ending position within the string to search. Note that the starting position can be to the right of the ending position, in which case the substring is search backwards.

char
is the character to search for, with two special wildcards: "%" matches any non-space in the substring, and "/" matches non-numeric, non-alphabetic character.

pos
will return the position of the matched character, or zero if there is no match within the specified substring.

NOECHO

xcall NOECHO {,channel}

As under AMOS, **NOECHO.SBR** may be used to disable terminal echo within a program. It is usually used in conjunction with GET or some other subroutine for terminal input, such as MicroSabio's INFLD in order to (a) allow character input, and (b) disable echo. Under A-Shell, NOECHO.SBR does not affect the true state of the terminal, but merely changes the internal operation of A-Shell itself. So, for example, if HOSTEX is used to execute a host command, that command will function in the same way whether or not the **NOECHO** subroutine had been used.

Under UNIX, a serial port may be opened for input, and then the **NOECHO** subroutine called with that channel specified, for example:

```
open #1,"/dev/tty1",input
xcall NOECHO,1
```

In this case, the true *host* mode of the port will be changed. Echo will be disabled, cooked mode processing will be disabled and so on, enabling all characters to be received unaltered from the port with the GET routine. This is particularly useful for communicating with external equipment such as modems, time record devices and so on.

If a port's mode is changed in this way, then it should be reset with **NOECHO.SBR** before closing the channel. Failure to do this will leave the port in its altered state even when A-Shell is exited.

ODTIM

```
xcall ODTIM, stringfmt, odate, otime, flags
```

ODTIM.SBR performs the reverse function of IDTIM.SBR, i.e. it converts an internal format date and/or time to a string format. It's formatting options are much more extensive, however, than the formats allowed by IDTIM.

Parameters

odate (numeric)
should contain the internal (aka "separated" – see IDTIM for details) date to be formatted. 0 is taken to mean the current date.

otime (numeric)
should contain the internal time to be formatted, with 0 indicating the current time.

flags (numeric)
must specify a sum of desired formatting options from the following table:

Value	Meaning
1	Omit date from output (ignore all other date related flags)
2	Output the day of the week
4	Use the full name of the day of the week; else use the first three characters
8	Output the month as a number (1-12) and ignore flag 16.
16	Output the full name of the month; else use the first character abbreviation
32	Output four digit year; else two digit year
64	Output the month first, then the day; else the day first, then the month
128	If 256 not specified, use spaces to separate the parts of the date, as in `17 Jul 2003`. If 512 is also specified (i.e. 128+256) then use separator character defined in the language definition file (LDF). Also see flags 128 and 8.
256	If 128 not specified, use slashes to separate the parts of the date, as in `7/17/03`. If 128 is also specified, use LDF separator character (see above).
512	Omit time from output (ignore all other time related flags)
1024	Omit seconds from the time output
2048	Use 12 hour time format with AM/PM; else use 24 hour format
4096	Do not output a separator between the hours and minutes (also see 131072)
8192	Use colon as the time separator; else use the character defined in the LDF
16384	Suppress leading zeroes from numeric portion of date.
131072	Do not output a separator between the minutes and the seconds
4194304	Do not output date punctuation
8388608	Do not output the day
16777216	Do not output the month

Value	Meaning
33554432	Do not output the year

Comments

If *flags* is set to 0, the output will appear like this: `17-Jul-03 13:15:58`. The special *flags* value –1 overrides the bit settings in the table and outputs a complete time and date string in the format: `Monday, July 17, 2003 01:15:58 PM` (with appropriate adjustments based on the language definition file).

PCKLST

Documentation update June 04; see "Update Notes" below. Also see XTREE, which provides an upward compatible, and much more powerful, version of PCKLST for GUI environments, and SBR=PCKLST_GUI in the Setup Guide.

```
xcall PCKLST, row, col, answer, array, maxcnt, prompt, exitcode
     {,strow, endrow, flags, file, mmoclr}
```

PCKLST.SBR simplifies the task of displaying pop-up pick lists of items to choose among. It is really just a front end to INMEMO, but does simplify things considerably. The idea is to call this routine instead of, or after a certain kind of exit from an ordinary INFLD or INPUT operation.

Parameters

row, col
are the locations where the ordinary field input would have taken place. The pick list box is drawn around the position where the field was or would have been (to give it the flavor of a drop-down box).

answer
On exit, *answer* is set to the Row # of the selection (first row is numbered 1). On input, *answer* specifies the default position of the selection bar.

array
is an array of strings containing the choices for the pick list. The end of the array is marked by the first null entry or *maxcnt*. Note that the array element size must be large enough to accommodate at least one trailing null on each element. Any text in an array element following a backslash will be treated as "hidden text", meaning that it will not be displayed (nor used in determining the box display width).

maxcnt (any numeric type)
is the maximum number of elements in the array.

prompt
contains the string to display in the top border of the box as a title or prompt. If the string contains an embedded CHR$(13), the remainder of the string will go on the bottom border of the box.

exitcode
will return a code indicating how the pick list was exited, with 0 indicating the normal exit via the ENTER key, and 1 indicating ESCAPE. See the Flags parameter below for enabling other exit keys.

strow, endrow (any numeric type)
if specified, determine the size and position of the box. Otherwise the box is likely to extend both above and below the row specified by the Row parameter, according to an internal (and top secret) set of rules. (Set to 0 as a placeholder if needed for subsequent parameters.)

flags (any numeric type)
may be used to enable additional exit keys beyond the normal ENTER (select) and ESC (abort without selection). Specify as the sum of zero or more of the following:

Value	Meaning
1	Allow F1-F16 (returning Exitcodes -1 thru -16)
2	Left arrow (Exitcode -40)
4	Enable Right arrow (Exitcode -41)
8	Enable Up arrow (Exitcode -42)
32	Enable TAB key (Exitcode -44)
64	Enable HOME key (Exitcode –45)
128	Enable END key (Exitcode –46)
256	Disable auto-shrink of box length to *maxcnt*
512	
1024	Time out after 200 seconds
2048	Causes the text version to use INMEMO's "fast menu" mode. In this mode, it is not necessary to hit ENTER to complete a selection; instead, you just need to enter a sufficient number of characters to uniquely identify a selection.

file

specifies a sequential file to be used instead of the Array parameter. When non-null, the interpretation of Array, Maxcnt, and Answer parameters changes as follows. Array becomes an ordinary string which is used to pass in the default selection (text) and to get back the item chosen by the user. Answer and Maxcnt are then ignored (with the actual contents of the file being examined to determine Maxcnt and the maximum width).

mmoclr

may be used to specify a set of colors for the various parts of the pick list display. It is equivalent to the INMEMO parameter of the same name, which is not surprising, since **PCKLST.SBR** is actually just a front-end to the vertical menu mode of **INMEMO.SBR**. The format of MMOCLR is:

```
MAP1 MMOCLR
    MAP2 BFCLR,B,1,-1           ! Border Foreground
    MAP2 BBCLR,B,1,-1           ! Border Background
    MAP2 TFCLR,B,1,-1           ! Text Foreground
    MAP2 TBCLR,B,1,-1           ! Text Background
    MAP2 AFCLR,B,1,-1           ! Arrow Foreground
    MAP2 ABCLR,B,1,-1           ! Arrow Background
    MAP2 PFCLR,B,1,-1           ! Prompt Foreground
    MAP2 PBCLR,B,1,-1           ! Prompt Background
    MAP2 WFCLR,B,1,-1           ! Warnings & messages Foreground
    MAP2 WBCLR,B,1,-1           ! Warnings & messages Background
    MAP2 SFCLR,B,1,-1           ! Orig. Status line Foreground
    MAP2 SBCLR,B,1,-1           ! Orig. Status line Background
    MAP2 RFCLR,B,1,-1           ! Reserved (unused) Foreground
    MAP2 RBCLR,B,1,-1           ! Reserved (unused) Background
```

The individual fields within MMOCLR are shown above initialized to -1, which is what you must do to get default colors, since color 0 is black. (Specifying MMOCLR with all the fields un-initialized will result in a not-very-interesting black on black display.)

Comments

The width of the box will be determined by considering the maximum length of the top and bottom prompts and the data items (not counting any hidden text).

Rather than setting colors in individual calls to PCKLST, it would be easier to set them globally in the LIB:INI.CLR file. See *Configuration* in the *A-Shell Setup Guide* for more details.

Update Notes: Build 863.3, 02 Feb 04

(Windows) PCKLST now uses a Windows-style listbox control if you are using one of the graphics terminal drivers (e.g. PCTDVG). The operation is similar to the text version of PCKLST. Current notable differences and limitations:

- The TFCLR (Text Foreground color) field within the MMOCLR structure is the only color parameter that is respected. The background color is hard coded to COLOR'WINDOWS (usually white).
- The top and bottom prompts are currently ignored. (They seem a bit odd looking in the Windows environment.)

Update Notes: Build 871, 07 Mar 04

PCKLST now allows the array argument to be passed as an unformatted variable, which gives you more flexibility in creating variable sized pick lists. Previously, for the array mode, you had to specify the first element, i.e. ARRAY(1), so that it the subroutine could determine the size of the array elements. Now, you can pass an unformatted variable which the array is overlaid upon, and PCKLST will determine the element size by looking at the spacing between the first and second elements in the array. For example:

```
MAP1 ARRAYX
   MAP2 ARRAY(50),S,13
xcall PCKLST,ROW,COL,ANSWER,ARRAYX,MAXCNT,PROMPT, &
   EXITCODE,STROW,ENDROW,FLAGS,"",MMOCLR
```

In order for this to work, there must be at least one trailing null terminating the ARRAY(1) element. Note that by using the overlay feature, you could support several array layouts with the same XCALL PCKLST, i.e.:

```
MAP1 ARRAYX2,@ARRAYX
   MAP2 ARRAY2(10),S,60
MAP1 ARRAYX3,@ARRAYX
   MAP2 ARRAY3(80),S,8
etc.
```

Update Notes: Build 883, 21 Apr 04

The *flags* argument now supports a new bit, 1024, which causes PCKLST it to time out in 200 seconds.

Update Notes: Build 884, 28 Apr 04

The *flags* argument now supports a new bit, 2048, which causes the text version to use INMEMO's “fast menu” mode. In this mode, it is not necessary to hit ENTER to complete a selection; instead, you just need to enter a sufficient number of characters to uniquely identify a selection.

PLYJOB

```
xcall PLYJOB, jobno, job'stats
```

PLYJOB.SBR is similar to GETJTB in that it returns most of the information related to your job in a single operation. The main reason it is included in addition to **GETJTB.SBR** is for PolyTrack programmers who may already be using the routine, and for anyone who may be using the MicroSabio routine JOBPAR.SBR (which is basically the same thing and can be aliased to PLYJOB using the alias facility in MIAME.INI). **PLYJOB.SBR** does have one feature that **GETJTB.SBR** does not: the ability to retrieve information about a job other than the current one. The amount of information that can be retrieved in this way is limited to job name, program name, ppn, and username, but that is still very useful in situations where you want to match a job number up with a job name.

The first parameter, JOBNO, must be mapped as a 2 byte Binary. If zero, information is returned about the current job, otherwise about the job whose number is specified. The second parameter is mapped as follows:

```
MAP1 JOB'TABLE                  ! 87 byte area
     MAP2 LOG'DEV,s,3           ! Device name
     MAP2 LOG'LUN,s,3           ! Logical unit
     MAP2 PROJECT,s,3           ! Project number (p,pn)
     MAP2 PROGRMR,s,3           ! Programmer number
     MAP2 PARTITION,s,6         ! Job name
     MAP2 PROG'ID,s,6           ! Program name
     MAP2 TERM'NAME,s,6         ! Trmdef name
     MAP2 TERM'DRVR,s,6         ! Terminal driver
     MAP2 INTERFACE,s,6         ! IDV name
     MAP2 JOB'NBR,s,3           ! Job number
     MAP2 JOB'TYPE,b,2          ! Job type flags
     MAP2 CMDFIL,b,2            ! Nonzero=command file
     MAP2 JOBPPN,b,2            ! P,PN as binary
     MAP2 JOBEXP,b,1            ! Job experience
     MAP2 JOBLVL,b,1            ! Job level
     MAP2 JOBUID,s,3            ! Numeric portion of username
     MAP2 JOBUSN,s,20           ! Job user name
     MAP2 JOBCPU,b,4            ! CPU time
     MAP2 JOBDSR,b,4            ! Disk reads
     MAP2 JOBDSW,b,4            ! Disk writes
     MAP2 JOBATT,s,6            ! Parent job name
     MAP2 JOBSTS,b,2            ! Job status
     MAP2 TRKVER,s,3            ! Tracker version [22C]
```

Several of the fields above are only applicable under the AMOS version of PLYJOB, including JOBCPU, JOBDSR, JOBDSW, JOBATT, JOBEXP, and JOBLVL.

PRTCHK

```
xcall PRTCHK, printer, rtncde
```

Under AMOS, **PRTCHK.SBR** may be used both to check for the existence of a printer, and to check whether a particular file is being printed. The A-Shell implementation contains only the first function, due to the limitations of either the spooling systems, or the interfaces to those spooling systems, of the operating systems on which it runs.

rtncde is a 6-byte floating point variable. In order to check for the existence of a printer it should be set to two before calling **PRTCHK**. If the printer exists, Rtncde will return as zero, otherwise it will be set to one. It should be noted that under both DOS/Windows and UNIX, a printer is considered defined if an initialization file exists in DSK0:[1,4] with the same name as the *printer* and extension .INI. The name of the A-Shell printers may differ from the names of the actual system printers to which they are assigned. This process is described in *Configuration in* the *A-Shell Setup Guide*.

> Apparently some versions of **PRTCHK.SBR** work the opposite way, returning 1 if the printer is found and 0 if it is not. To get this behavior, add SBR=PRTCHK1 to the MIAME.INI file.

PUTBYT

```
xcall PUTBYT, channel, buffer, bytes
```

PUTBYT.SBR writes raw bytes to a sequential output file.

Parameters

channel (numeric)
is the channel number of the file, which must be open for output or append.

*buffer (*unformatted)
supplies the data to be written.

bytes (numeric)
specifies the number of bytes to write (which must be less than or equal to the size of *buffer*).

Comments

```
xcall PUTBYT, channel, buffer, bytes
```

is equivalent to

```
PRINT #channel, buffer[1,bytes]
```

Also see GETBYT.

RXTERM

```
xcall RXTERM, exitcode
```

RXTERM.SBR (Windows only) looks in the type-ahead buffer to see if a function key or Control-C character has been entered, and if so, returns the INFLD-compatible *exitcode*. This can be useful to short-circuit some program actions and provide immediate response to a function key command.

SEND

```
xcall SEND, job, string {,options {,status}}
```

SEND.SBR provides the same functionality at the programmer level that **SEND.LIT** does at the command prompt level.

Parameters

job
: may be specified as B,1 in which case it is interpreted as the target job number, or S,6 in which case it is the target job name.

string
: is the text message (one line) to send.

options (optional)
: may be set to 0 (normal, i.e. display the message wherever the cursor happens to be on the target terminal), 1 (top status line), or 2 (bottom status line).

status
: if specified, will return 0 if the message was successfully sent.

Comments

Under UNIX/Linux, there are security limitations on who can send messages. See the discussion of privilege settings under the Installing A-Shell/UNIX in the *A-Shell Setup Guide*, or under ITC, for more details.

Under Windows, there are no privilege issues but there may be a delay of several seconds before the message is delivered. See the documentation for the IJCFREQ parameter in the MIAME.INI section of the *A-Shell Setup Guide.*

SERCH

```
xcall SERCH, ch, rec, key, stpos1, enpos1, bsend, bsmid, srcctl,
     srcopt, rcnost, rcnoen, stpos2, enpos2, stpos3, enpos3
```

SERCH.SBR is fully compatible with the AMOS version. It is used for searching index files that consist of a sorted area and an unsorted "overflow" area. (The sorted area is searched using a binary search routine, while the overflow area is searched sequentially.) At best, binary searching is nowhere near as efficient as ISAM index lookups, and this discrepancy gets rapidly worse as the size of the overflow area increases, so it is not a highly recommended indexing technique. Nevertheless it is quite common, due to its use in the old AlphaAccounting programs and concerns in the early days about the reliability of ISAM.

Parameters

ch (any numeric type)
is the open file channel of the index file.

rec
is an unformatted area for the located index record to be returned in.

key
is the string key to search for. This can consist of up to 3 parts (corresponding to the *stpos1*, *enpos1*, *stpos2*, *enpos2*, *stpos3* and *enpos3* locations in the index record), all of which must be concatenated together.

stpos1, enpos1, stpos2, enpos2, stpos3, enpos3 (any numeric types)
specify the starting and ending position of up to 3 segments of the index record which make up the key to search for.

bsend (numeric)
is the number of records in the sorted portion of the file (which must begin at the first record in the file).

bsmid (F,6)
will return the record number of the located index record.

srcctl (F,6)
returns a completion code (0=found, 1=not found)

srcopt (numeric)
specifies the type of search to perform:

Value	Search Type
1	Display "Please Wait" on line 12 of the screen, perform a binary/sequential search, clear the message, and return.
2	Perform a binary/sequential search, clear line 12 of the screen, and return
3	Display the wait message, perform a binary/sequential search, and return
4	Perform a binary/sequential search and return (without any display)

Value	Search Type
5	Perform a sequential search only, starting with the record number specified by *bsmid*.
6	Perform a binary/sequential search which guarantees the locating of the first of a series of identical keys. (The other binary search options do not guarantee this.)

By "binary/sequential" in the table above, we mean that it first performs a binary search on the sorted portion of the index file, and if the record is not found, it then performs a sequential search on the overflow area.

rcnost, rcnoen
mark the starting and ending positions within the record that are used to indicate a deleted record. This area must be at least 2 bytes long, and each of the bytes must contain binary zero. (You can create such a binary field by setting a variable of type F or B to zero, in which case all of the bytes of the variable will contain 0, or by setting a string variable to "".)

Comments

The speed of **SERCH.SBR** can be increased dramatically by using one of the techniques described in the discussion of ASFLAG. It also goes without saying that you should sort the file as needed to keep the overflow area from growing very large.

SETJTB

```
xcall SETJTB, job-table-variable
```

SETJTB.SBR is the converse of GETJTB and operates on the same format structured variable. It writes back each field into the job control block (with the exception of the job number, which cannot be changed), and thus may be used to change PPNs and devices, or to change the job name.

Comments

Care must be taken when using **SETJTB**, as it does not perform any checking on the values within the job table variable, and it is possible to store invalid information in your job control block.

Note that the PPN is represented both in its binary and string forms in *job-table-variable*. The current PPN will be set to the string values only if both binary fields contain zero. (Using the string fields is preferable, since they support the full range of 000-999, while the single byte binary versions are limited to octal values 0-377.)

SIZE

```
xcall SIZE, fspec, bytes
```

SIZE.SBR returns the size of the specified file in bytes.

Parameters

fspec (string)
is specification of the file (in AMOS or native format).

bytes (F,6)
returns the number of bytes. If the file does not exist, -1 is returned. A return value of 0 indicates that the file exists but contains no data.

SORTIT

Documentation update June 04; see "Update Notes" below.

```
xcall SORTIT, array, reccnt, recsiz, k1siz, k1pos, k1ord, k2siz,
      k2pos, k2ord, k3siz, k3pos, k3ord, k1typ, k2typ, k3typ
```

SORTIT.SBR is basically the same as BASORT, except it is used to sort arrays in memory rather than files on disk. The arguments are all analogous to those of **BASORT** (and therefore, hopefully, self-explanatory) and like BASORT, the default key type is string, and the parameters for the second and third keys (and types) are optional. The *array* argument is typically specified as the name of a MAP1 variable, under which the array is mapped, as shown here:

```
MAP1 ARY
   MAP2 SRTARY(50)
      MAP3 NAME,S,20
      MAP3 AGE,B,2
      etc.
```

Update Notes: Build 858.9 - 12 Jan 04

SORTIT.SBR now supports up to 6 sort keys:

```
xcall SORTIT, array, reccnt, recsiz, k1siz, k1pos, k1ord, k2siz,
      k2pos, k2ord, k3siz, k3pos, k3ord, k1typ, k2typ, k3typ, k4siz,
      k4pos, k4ord, k5siz, k5pos, k5ord, k6siz, k6pos, k6ord, k4typ,
      k5typ, k6typ
```

Everything after *k1ord* is optional; if you specify one of the keys, you must specify the *k?siz*, *k?pos*, and *k?ord* arguments, but may omit the *k?typ* argument (in which case it defaults to 0 or string.)

SPOOL

SPOOL.SBR is handled by using the ALIAS facility in the A-Shell configuration file to redirect it to **EZSPL.SBR**, which supports a superset of the normal SPOOL.SBR capabilities. See the discussion on EZSPL for more details.

STRIP

```
xcall STRIP, strvar
```

STRIP.SBR removes trailing blanks from the specified string. Note that it does not remove TABs or any other control characters, just spaces, and just those that are at the trailing end.

Also see TRIM, which trims both leading and trailing spaces, and has an option to remove unprintable characters as well.

STRTOK

Documentation updates: **STRTOK.SBR** was added to A-Shell in build 854, 21 Nov 03. Examples added November 04.

```
xcall strtok, op, string, fdelim, fld11 {, rdelim, fld2, ...fldn}
```

STRTOK.SBR is both convenient and very efficient (compared to the equivalent logic in Basic) for parsing string data, but may take a little practice to get used to. See below for a couple of real examples of using it.

STRTOK.SBR parses the STRING variable according to the delimiters specified, and returns one or more fields in the FLD1...FLDn variables. You can call it once for each field (and change the field delimiter characters as you go), or have it return several parsed fields in one call, ending either when it hits a delimiter in the RDELIM field or runs out of STRING or of FLDx parameters.

Two examples, Almost Comma Delimited and Large Packets, Multiple Delimiters are provided below.

Parameters

op (numeric)
should be set to 0 for the initial call, and 1 for subsequent calls using the original STRING. (It will be updated automatically from 0 to 1 to make this easy.)

string (string)
is the string to be parsed. Must be null terminated! STRING will be modified by the routine, which will replace the delimiter characters with null bytes.

fdelim (string)
is a list of one or more field delimiter characters.

rdelim (string)
is a list of one or more "record" delimiter characters.

(The XCALL will terminate when it hits one of these, whereas it will keep going after each field delimiter until all the FLDx parameters are used up.) Parameter is irrelevant if you just want to get one field. Even in the case of multiple fields, you can set it to "". Note, however, that if specified, it will be returned updated with the actual last delimiter character processed (i.e. the one that terminated the xcall). This can be very useful when there are more than one possible delimiter, or when you want to determine whether you got a complete record or just ran out of FLDx parameters. If you don't care about record delimiters, then specify it as a literal "" (in which case it can't be updated) or remember to clear it prior to each XCALL STRTOK. (Otherwise, it will get updated to match the field delimiter. The XCALL should be smart enough to ignore record delimiters that are also field delimiters, but it could nonetheless lead to confusion.)

Note that prior to Build 905.3, rdelim was assumed to be 2 or more bytes and the second byte was getting cleared, which would have clobbered the next variable if rdelim only mapped as one byte.

fld1 ...fldn (strings)
will return the parsed fields or tokens from STRING, according to the specified delimiters.

Almost Comma Delimited

Ordinary comma delimited data is easy to parse with INPUT CSV, but what do you do when the comma delimited file contains commented lines? A good example is the A-Shell INI.CLR file, which looks something like this:

```
; Color Definitions (for a primarily blue background)
; (Color legend: 0=blk, 1=wht, 2=blu, 3=mag, 4=red, 5=yel,
;   6=grn, 7=cyn)
; VUECLR=override?,edit fg,edit bg, command fg, command bg,
;   help fg, help bg, status/info fg,bg
; (Use the following for a reasonable scheme using
;   blue background...)
VUECLR=Y, 1,2, 5,2, 7,2, 6,2
; (Use the following to disable VUE colors,
;   i.e. use current colors only...)
;VUECLR=Y,-1,-1,-1,-1,-1,-1,-1,-1
; EZCLR=override?,text fg,bg, border fg,bg, cmd fg,bg,
;   sts fg,bg, help fg,bg, highlight fg,bg, menu fg,bg,
;   brief menu fg,bg
EZCLR=Y, 1,2, 1,2, 5,2, 0,9, 1,3, 1,4, 5,2, 7,2
; INFCLR=override?,display fg,bg, edit fg,bg,
;   negative fg,bg, update fg,bg, message fg,bg,
;   original msg line fg,bg, forms fg,bg
INFCLR=Y, 5,2, 1,10, 4,2, 7,2, 7,2, 1,2, 0,2
; MMOCLR=Y,border fg,bg, text fg,bg, arrows fg,bg,
;   prompt fg,bg, status line fg,bg, protected fg,bg
MMOCLR=Y, 6,2, 5,10, 6,2, 7,2, 1,2, 7,2
; SCNCLR=override?,screen text (fg), screen background
;  (The following sets up white on blue)
SCNCLR=Y,1,2
```

This seems like a simple enough format, but the unpredictable presence of the commented lines (marked by a semicolon), and the equal sign make it difficult or impossible to use INPUT CSV. **STRTOK** comes in handy here because it can be used on a string, allowing you to first input the entire line to check for commented out lines, then pass it to **STRTOK**. Also, it eliminates the need to deal with special tests for prematurely terminated lines, lines with comments in them, uneven spacing, etc. The following code parses the input, putting into the MAP structures:

```
MAP1 VUECLR'OVR,S,1
MAP1 VUECLR(4,2),S,1   ! 4 pairs
MAP1 EZCLR'OVR,S,1
MAP1 EZCLR(8,2),S,1    ! 8 pairs
MAP1 INFCLR'OVR,S,1
MAP1 INFCLR(7,2),S,1   ! 7 pairs
MAP1 MMOCLR'OVR,S,1
MAP1 MMOCLR(6,2),S,1   ! 6 pairs
MAP1 SCNCLR'OVR,S,1
MAP1 SCNCLR(2),S,1     ! 1 pair
MAP1 PLINE,S,100
MAP1 STATE,F,6,1       ! 1)VUECLR, 2)EZCLR, etc.
MAP1 RDELIM,S,2
MAP1 FDELIM,S,3,"=,"
MAP1 OP,B,1,0

   OPEN #1, "LIB:INI.CLR", INPUT
LOOP:
   INPUT LINE #1, PLINE
   IF EOF(1)=1 AND PLINE="" GOTO PREMATURE'END
   xcall TRIM,PLINE,1
```

```
    IF PLINE="" OR PLINE[1,1]=";" GOTO LOOP

    OP = 0
    ON STATE CALL PVUE,PEZ,PINF,PMMO,PSCN
    IF STATE < 5 GOTO LOOP

PREMATURE'DONE:
    PRINT "PREMATURE END OF FILE"

DONE:
    CLOSE #1
    END


PVUE:    ! parse VUECLR line
    RDELIM = ";" + CHR(10)
    xcall STRTOK,OP,PLINE,FDELIM,VUECLR'OVR,RDELIM, &
        VUECLR(1,1),VUECLR(1,2),VUECLR(2,1),VUECLR(2,2), &
        VUECLR(3,1),VUECLR(3,2),VUECLR(4,1),VUECLR(4,2)
        STATE = STATE + 1
    RETURN

PEZ:    ! parse EZCLR line
    RDELIM = ";" + CHR(10)
    xcall STRTOK,OP,PLINE,FDELIM,EZCLR'OVR,RDELIM, &
        EZCLR(1,1),EZCLR(1,2),EZCLR(2,1),EZCLR(2,2), &
        EZCLR(3,1),EZCLR(3,2),EZCLR(4,1),EZCLR(4,2), &
        EZCLR(5,1),EZCLR(5,2),EZCLR(6,1),EZCLR(6,2), &
        EZCLR(7,1),EZCLR(7,2),EZCLR(8,1),EZCLR(8,2)
    STATE = STATE + 1
    RETURN

PINF:    ! parse INFCLR line
    RDELIM = ";" + CHR(10)
    xcall STRTOK,OP,PLINE,FDELIM,INFCLR'OVR,RDELIM, &
        INFCLR(1,1),INFCLR(1,2),INFCLR(2,1),INFCLR(2,2), &
        INFCLR(3,1),INFCLR(3,2),INFCLR(4,1),INFCLR(4,2), &
        INFCLR(5,1),INFCLR(5,2),INFCLR(6,1),INFCLR(6,2), &
        INFCLR(7,1),INFCLR(7,2)
    STATE = STATE + 1
    RETURN

PMMO:    ! parse MMOCLR line
    RDELIM = ";" + CHR(10)
    xcall STRTOK,OP,PLINE,FDELIM,MMOCLR'OVR,RDELIM, &
        MMOCLR(1,1),MMOCLR(1,2),MMOCLR(2,1),MMOCLR(2,2), &
        MMOCLR(3,1),MMOCLR(3,2),MMOCLR(4,1),MMOCLR(4,2), &
        MMOCLR(5,1),MMOCLR(5,2),MMOCLR(6,1),MMOCLR(6,2), &
        MMOCLR(7,1),MMOCLR(7,2)
    STATE = STATE + 1
    RETURN

PSCN:    ! parse SCNCLR line
    RDELIM = ";" + CHR(10)
    xcall STRTOK,OP,PLINE,FDELIM,SCNCLR'OVR,RDELIM, &
        SCNCLR(1),SCNCLR(2)
    STATE = STATE + 1
    RETURN
```

The example above was complicated by the fact that every line of the file contained a different format and needed to be put into different variables. So we needed a lot of individual STRTOK commands. But we didn't need any complicated substring processing – it was all very straightforward.

Large Packets, Multiple Delimiters

The next example is perhaps more typical, taken from a real-world data stream arriving over a TCP socket. The data represents the answer to an auto parts query, and is made up of a list of auto makes. Within each make is a list of models. And within each model is a list of years. The model records are terminated with a "|", the make records are terminated with ~, and the individual fields are terminated with semicolons. So a block of data may look something like this:

```
Ford; Mustang; 1975-79; 1985-88; 1999-2004|Pinto; 1968,1971,1980-85,1992|Explorer;
   1992,1994-96; 1998-2004|Expedition; 2001; 2003~Chevrolet; Impala; 1957-61;
   1965-70; 1976,1979|Camaro; 1962; 1964; 1966; 1968; 1975~Mazda; RX7; 1985-91;
   1994|Miata; 2004~
```

Our program to parse the data looks like this:

```
MAP1 PARAMS
   MAP2 MAKES(20)              ! Up to 20 makes
   MAP3 MKNAME,S,10            ! name of make
   MAP3 MODELS(20)             ! up to 20 models per make
      MAP4 MODNAME,S,10       ! model name
      MAP4 YEARS(30),S,10     ! up to 30 years per model
MAP1 PARAMS$,S,124200,@PARAMS   ! for easy init of all

MAP1 INFO,S,10000           ! block of data
MAP1 RDELIM,S,2,""
MAP1 FDELIM,S,1,";"         ! field delimiter
MAP1 OP,B,1
MAP1 MKX,F,6                ! index to MAKES()
MAP1 MDX,F,6                ! index to MODELS()
MAP1 YX,F,6                 ! index to YEARS()

      !   <receive data from socket into INFO>
      INFO = "Ford;Mustang;1975-79;1985-88;1999-2004|" &
      + "Pinto;1968;1971;1980-85;1992|Explorer;1992;" &
      + "1994-96;1998-2004|Expedition;"2001;2003~Chevrolet;"
      + "Impala;1957-61;1965-70;1976,1979|Camaro;"
      + "1962;1964;1966; 1968;1975;1976; 1977;1978;"
      + "1980;1982;1984;1986; 1988;1990;1992;" &
      + "1994;1996;1998;1999;2000;2002;2004~" &
      + "Mazda;RX7;1985-91;1994|Miata;2004~"

      PARAMS$ = ""   ! Clear entire param array
MAKE'LOOP:      ! input one 'make' record per loop
   MKX = MKX + 1
   RDELIM = ""
   MAKE$ = ""   ! clear entire MAKES(MKX)

   xcall STRTOK,OP,INFO,FDELIM,MKNAME(MKX)
   IF MKNAME(MKX)="" GOTO DONE
   MDX = 0

   MODEL'LOOP:     ! input models, one per loop
   MDX = MDX + 1
   RDELIM = "~|"   ! could end on either delimiter

   xcall STRTOK,OP,INFO,FDELIM,MODNAME(MKX,MDX),RDELIM, &
      YEARS(MKX,MDX,1),YEARS(MKX,MDX,2),YEARS(MKX,MDX,3), &
      YEARS(MKX,MDX,4),YEARS(MKX,MDX,5),YEARS(MKX,MDX,6), &
      YEARS(MKX,MDX,7),YEARS(MKX,MDX,8),YEARS(MKX,MDX,9), &
      YEARS(MKX,MDX,10),YEARS(MKX,MDX,11),YEARS(MKX,MDX,12), &
```

```
   YEARS(MKX,MDX,13),YEARS(MKX,MDX,14),YEARS(MKX,MDX,15)

! check what delimiter we ended with...
IF RDELIM="~" GOTO MAKE'LOOP   ! get next make
IF RDELIM="|" GOTO MODEL'LOOP  ! get next model
IF RDELIM<>";" GOTO ERROR      ! unexpected delimiter

! if last delimiter ; get more fields - note we have
! a limit of 30 params so we couldn't do them all in
! one xcall
RDELIM = "~|"   ! could end on either delimiter
xcall STRTOK,OP,INFO,FDELIM,YEARS(MKX,MDX,16),RDELIM, &
   YEARS(MKX,MDX,17),YEARS(MKX,MDX,18),YEARS(MKX,MDX,19), &
   YEARS(MKX,MDX,20),YEARS(MKX,MDX,21),YEARS(MKX,MDX,22), &
   YEARS(MKX,MDX,23),YEARS(MKX,MDX,24),YEARS(MKX,MDX,25), &
   YEARS(MKX,MDX,26),YEARS(MKX,MDX,27),YEARS(MKX,MDX,28), &
   YEARS(MKX,MDX,29),YEARS(MKX,MDX,30)GOTO MAKE'LOOP
GOTO MAKE'LOOP
```

Note that STRTOK does not clear the unused fields, so it is up to the program to reset them in advance. In the program above, although this is not a problem since we only use the arrays once, we use a string overlay to clear the entire 3 level array in one step just to illustrate the general case.

SUBMIT

```
xcall SUBMIT, rtnvar, stdin, stdout, stderr, ashell'cmd, arg1, arg2
     {,...argn}
```

SUBMIT.SBR allows you to submit a background task to be executed. It is only supported under UNIX versions of A-Shell, and it is primarily considered an internal routine (created to support SUBMIT.LIT). So if you choose to use it, you would be well advised to create a BASIC wrapper for it to make it easier to adapt if we decide to modify the parameters (to add functionality) later.

Parameters

rtnvar (B,2)
will return with the process id number of the child process created to run the task.

stdin
is the (AMOS) filespec of the AUI_CONTROL file containing the input to be forced to the process.

stdout, stderr
are the (AMOS) filespecs to send normal and error output messages to. To mimic the AMOS LOG file behavior, set them both to the same name (typically with an extension of LOG and a base name either matching the control file or the job name).

ashell'cmd
and the subsequent arguments are used to launch the new copy of A-Shell which will run the background process. For example:

```
xcall SUBMIT, RC, "CMD:TEST.CTL", "TEST.LOG", "TEST.LOG", "ashell"
```

Comments

Note that there are a number of differences between **SUBMIT.SBR** under A-Shell and submitting a task to the AMOS Task Manager. First, with **SUBMIT.SBR**, there is no task manager; instead, the process runs as a child to your current process. Second, there is no queue; the task starts executing immediately (and there is no particular limit to how many tasks you can submit this way with overlapping execution). Third, none of the AMOS embedded control file directives (which contain the $ symbol) are supported under A-Shell. Fourth, although you can use the SUBMIT.LIT program to check on the status of background tasks (and even kill them), under A-Shell, you can only do so for tasks which *you* have submitted (unless you are the Superuser).

SWPSBR

```
xcall SWPSBR, function, {parameters dependent on function}
```

SWPSBR.SBR performs various functions related to the Swap multi-session and screen-tracking system available as a third-party add-on in the AMOS world. A-Shell doesn't support Swap per se, since it has its own screen tracking system (TRACKER) and multi-session system (either PolyShell or using multiple Windows instances), but it does support most of the SWPSBR functions, which are listed below. Note that unless you already have a lot of code using these functions, you would be better off to use MSBOXX, which can do all of these things and which is both more powerful and better supported.

```
xcall SWPSBR, 0, active$
```

Function 0 tests whether the Swap system is active. A-Shell returns "Y" in the *active$* parameter.

```
xcall SWPSBR, 1, screen

xcall SWPSBR, 1, srow, scol, erow, ecol, screen
```

Function 1 restores the previously saved window or entire screen. With Swap, the screen parameter was a buffer to hold the screen contents, but it is ignored with A-Shell, which has an internal storage scheme. (You can map it as X,1 to save space.) However, A-Shell's scheme acts like a stack, so you must save and restore in LIFO order (i.e. last item saved will be the first item restored.) The first format above restores the entire screen. The second format restores a previously saved partial screen.

> You can also restore an entire screen with TAB(-1,203), assuming it has previously been saved with TAB(-1,202) or an equivalent. You can also save and restore partial screens using MSBOXX.

```
xcall SWPSBR, 3, screen

xcall SWPSBR, 3, srow, scol, erow, ecol, screen
```

Function 3 saves an entire screen (first format) or partial screen (second format). See Function 1 above for the restore operation.

```
xcall SWPSBR, 7, srow, scol, erow, ecol, siz
```

Function 7 was used to establish the size needed to store the specified screen area. The call is simply ignored by A-Shell, since you don't need to provide any storage space for the save and restore operations.

```
xcall SWPSBR, 8, srow, scol, erow, ecol
```

Function 8 was used to restore attributes on the right side of the box. It is not implemented by A-Shell. You don't need this capability with mode emulations (such as Windows, VT100, etc.) and if you really need it, you can get it by using MSBOXX instead using the BOX_FAO option, or via Function 128.

```
xcall SWPSBR, 9, srow, scol, erow, ecol
```

Function 9 clears the specified box. It is equivalent to the MSBOXX option BOX_ERA.

```
xcall SWPSBR, 128, srow, scol, erow, ecol, flags$
```

Function 128 draws a box, with various options determined by the contents of the *flags$* parameter. The options are listed in the table below, and may be strung together using commas. For example, "T,C,B,O=2,I=3."

Flags$	Description
B	Draw a border around the box. (Equivalent to MSBOXX option BOX_BDR.)
C	Clear interior of box. (Equivalent to MSBOXX option BOX_ERA.)
T	Seal up field attributes. (Equivalent to MSBOXX option BOX_FAO.)
I=<text fg>	Sets the text (interior) foreground color. For example, I=1 would set it to white. (Equivalent to the IBOX'FG parameter in MSBOXX.)
O=<border fg>	Sets the border (outside) foreground color. For example, O=2 would set it to color 2 (blue). (Equivalent to the BRDR'FG parameter in MSBOXX.)

```
xcall SWPSBR, 129, screen, fspec$
```

Function 129 saves the current screen, VUEs the specified file, then restores the screen. As with the other functions involving the Screen parameter, Screen is ignored by A-Shell, which instead uses EZTYP to save the screen, display the file, and restore the screen on exit.

TCPCLI

This subroutine has been superceded by TCPX; it is supported and documented for historical reasons only. Unless you are running an old version of A-Shell (prior to build 855 of November 2003—in which case you should update!), you should use TCPX.

```
xcall TCPCLI, code, status, buffer, sockport, hostname {,flags}
```

TCPCLI.SBR provides a way for a client process to communicate with a server process over TCP/IP sockets. It is specifically designed to interface easily with another A-Shell server process which is running TCPSRV, but it is flexible enough to be used with a wide variety of server processes.

Parameters

code (any numeric type)
Specifies operation, from choices below:

Value	Description
1	Connect to TCPSRV server (same as 9 but waits for a response)
2	Write contents of BUFFER to socket.
4	Read data from socket into BUFFER
6	Close socket
7	Check if socket has data to be read (returns STATUS=1 if so)
8	(UNIX only) Same as 7 but returns STATUS=# bytes available to read
9	Connect to server; same as 1 but does not wait for a response (works for any server, including TCPSRV)

status
Return status. <0 for errors (-errno), >0 indicates number of characters written or read.

buffer (S or X)
Read/write buffer (max 4096 bytes)

sockport (F,6)
For connect, caller must supply port # to connect to. On return, this will contain the connected socket number, which must then be passed to all the other calls for this session. If the connect operation returns <0, this indicates an error (with the number indicating which of the several internal steps failed).

hostname
On connect, must contain the host name or IP address of the server.

flags (optional; F,6)
On the connect calls (1 & 9), set to 1 for a blocking connection, or 0 for a non-blocking connection. (Default is blocking.) On the check operation, set to # seconds to wait before returning if no bytes are available. (Fractional seconds are ok; default is 0 if parameter not specified.) On the read call

specifies (if non-zero) the maximum # of bytes to read (default is the size of BUFFER). On the write call, specifies (if non-zero) the number of bytes to write.

Comments

Sockets are not automatically closed when a BASIC program terminates, so it is important that you add a socket close command to your error trap. Otherwise you may not be able to reconnect to the socket.

You can translate the system error codes returned as negative numbers in the STATUS parameter by using XCALL MIAMEX,86,ABS(STATUS),MESSAGE$

The client connect calls will fail if the server is not already waiting for a connection. (In contrast, the server side the corresponding "accept connection" (Code 1) call using **TCPSRV**, will wait indefinitely for a client to make a connection.)

The number of bytes to read or write (as specified by FLAGS) will be trimmed to the smaller of the size of BUFFER and 4096 bytes. The one exception is that if the BUFFER variable is subscripted, then FLAGS may be as large as 4096. (This overcomes the obstacle when using an array caused by the fact that the subroutine calling mechanism reports the size of just one element, rather than the size of the entire array.)

Opcode 9 is recommended in place of Opcode 1, even when you know the server is running **TCPSRV**. The only difference is that Opcode 1 does an automatic read after connecting. It is more standard and flexible to issue a separate read to retrieve the server's ACK message (if any).

TCPSRV

This subroutine has been superceded by TCPX; it is supported and documented for historical reasons only. Unless you are running an old version of A-Shell (prior to build 855 of November 2003—in which case you should update!), you should use TCPX.

```
xcall TCPSRV, code, status, buffer, sockport {,flags}
```

TCPSRV.SBR provides a way to create server processes that support client requests via a socket interface. Similar to TCPCLI, and in fact designed specifically to integrate easily with it, **TCPSRV.SBR** is nevertheless flexible enough that it can be used to communicate with a wide variety of client processes over TCP/IP sockets. For example, you could use it to create a server that responded to web queries submitted from a CGI process.

Parameters

code (any numeric type)
Specifies operation, from choices below:

Value	Description
1	Create listening socket on port specified by SOCKPORT, and wait for a client to connect. If BUFFER parameter is non-null, then after the connection is accepted, the first 32 bytes of the BUFFERDATA parameter will be sent to the client as an acknowledgment message. On successful return, SOCKPORT is set to the connected socket, which must be specified to all the other calls for this socket.
2	Write contents of BUFFER to socket.
4	Read data from socket into BUFFER
6	Close socket
7	Check if socket has data to be read (returns FLAGS=1 if so)
8	(UNIX only) Like CODE 7 except returns the number of bytes available to read.

status (F,6)
Return status. <0 for errors (-errno), >0 indicates number of characters written or read.

buffer (S or X)
Read/write buffer (max 4096 bytes)

sockport (S or X)
(F,6) For connect, caller must supply port # to connect to. On return, this will contain the connected socket number, which must then be passed to all the other calls for this session. If the connect operation returns <0, this indicates an error (with the number indicating which of the several internal steps failed).

flags (optional; F,6)
On the connect operation, may contain 1 for a blocking connection, or 0 for a non-blocking connection. (Default is blocking.) On the check operation, contains the number of seconds to wait before returning if no characters are available. (Fractional seconds are ok; default is 0 if parameter not specified.) For the write operation, it specifies the number of characters to write. (Default is size of the BUFFER parameter if FLAGS not specified.) For the read operation, if non-zero, specifies the maximum number of bytes to read.

Comments

Note that only one server process can wait for connections on a particular port number for a single machine. So it is important that the server do its job and close the socket quickly if there is a possibility of multiple clients making frequent requests. Typically, this kind of single-threaded server works best for quick and simple lookup operations, such as price lookups or inventory status checks.

See the sample program TCPTST.BAS, available on our web server, for a working example, and see A-Shell Technical Note 1013 for a more detailed discussion of socket programming, including some tips on how to create a server that can handle multiple simultaneous connections.

TCPX

```
xcall TCPX, op, status, data, sockport {,flags{,timer{,hostname}}}
```

TCPX.SBR was added to A-Shell in build 855, 28 November 2003. It is a consolidation of older subroutines **TCPCLI** and **TCPSRV**, which are now obsolete. The old calling interfaces are still supported, but should be replaced with TCPX when convenient. See the documentation on TCPSRV and TCPCLI for additional background information on TCPX.

Parameters

op (numeric)
indicates operation, per the following list:

Value	Operation
1	(server) wait for client connection
2	write DATA to socket
4	read DATA from socket
6	close socket
7	check if data avail to read (return 1 in FLAGS if so)
8	check how much data avail to read (return # bytes in FLAGS)
9	(client) connect to generic server
10	(client) (equivalent to old TCPCLI opcode 1; now deprecated)
11	return system error message corresponding to STATUS

status (float)
Returns status of operation. <0 = -errno. >0 indicates number of bytes read or written.

data (string, unformatted, or array)
Packet of data to read or write. Note changes: there is no longer any hardcoded limit on the packet size; however the OS may impose its own limits.

sockport (numeric)
On connect or accept (OPs 1,9,10) must supply the port # to listen on or connect to; returns the connected socket #, which must be supplied to all other calls (except OP 11).

flags (numeric, optional)
Usage varies with *op*. Note changes on read and write operations: you may now specify number of bytes to read/write and a timeout value in TIMER to ensure whole logical packet read/writes in blocking connections, without risking being stuck for an inordinate amount of time if other end malfunctions.

Value	Operation
1	set=1 for blocking mode

Value	Operation
2	specifies the number of bytes to write. Must be <= size of DATA, unless DATA is subscripted, in which case we just trust the FLAGS value. 0 (or FLAGS not specified) means write entire DATA block. If FLAGS non-zero, we set the send buffer low water mark so that we don't attempt operation unless buffer has sufficient space. (This is one case where you might have a problem if you use a packet size larger than the OS supports.)
4	specifies the number of bytes to read. 0 means read whatever is available (from 1 byte to size of DATA). For blocking connections, subroutine will wait until the required number of bytes is read (up to TIMER limit). For non-blocking connections, this size is only an upper limit; the routine will return as soon as the interface indicates there is data available (even if not the requested number of bytes.)
6	Initiate a "graceful" background shutdown. Call returns immediately; OS tries to gracefully close the connection. If other end is waiting on input (in OP 4), it will receive an error.
7	If TIMER specified, FLAGS is ignored and TIMER is used. If TIMER not specified, FLAGS specifies how long to wait before returning STATUS 0 if no data avail. If FLAGS is F,6 then it is interpreted as fractional seconds; otherwise it is interpreted as milliseconds.
8	FLAGS not used.
9	(client) connect to generic server
10	(client) (equivalent to old TCPCLI opcode 1; now deprecated)
11	FLAGS not used.

timer (numeric, optional)

For OP 7, number of milliseconds to wait before returning if no data avail. For reads and writes on blocking connections, number of milliseconds to wait before returning if full operation cannot be satisfied.

hostname (string)

Used only on the connect OP; supplies the name of the host where the server is running. As a convenience for backwards compatibility with TCPCLI/TCPSRV, it can be the 5th parameter (if 6 parameters specified and 6th is numeric), or the last parameter (with 5, 6, or 7 parameters specified).

Comments

As a convenience for backwards compatibility with **TCPCLI** on the connect OP, FLAGS can be the 6th parameter if the 5th parameter is a string (i.e. **TCPX** is called with the **TCPCLI** syntax:

```
xcall TCPX, op, status, data, sockport, hostname, flags
```

Update notes: Build 908.1 - 17 Nov 2004

TCPX has been enhanced to allow for better implementation of an AlphaBasic server that receives rapid connection requests, whether they are handed off to spawned children or by server itself. The basic problem prior to this update was that the listening socket was closed as soon as a connection was accepted. Even if the server quickly processed the request and returned to accept another connection, any connection requests arriving while the listening socket was closed would have been rejected. The new feature allows the listening

socket to be kept open independently of the connection socket, and the size of the queue has been increased from 5 to 64.

To review, the syntax of TCPX.SBX is:

```
xcall TCPX,op,status,data,sockport{,flags{,timer{,hostname}}}
```

On the ACCEPT OP (1), the FLAGS argument can be a reasonable combination of following (all but the first one are new in this update):

Flag Name	Opcode	Meaning
TCPXFLG'BLOCK	1	Establish connection being accepted as blocking.
TCPXFLG'LISTEN	4	Causes TCPX to return with the listening SOCKET (in SOCKPORT) w/o waiting to accept a connection.
TCPXFLG'ASYNC	8	Accept connection on previously opened listening socket. (SOCKPORT must be the listening socket.)
TCPXFLG'KEEPLISTEN	16	Combined with TCPXFLG'ASYNC to keep the listening socket open after accepting the connection.

Normally (when none of the last 3 flags are set), the ACCEPT operation will open a listening socket on the port specified by SOCKPORT, and then wait for a connection to be accepted. At that point, it closes the listening socket and returns the connection socket in SOCKPORT to the caller.

If the TCPXFLAG'LISTEN flag is specified, then it returns with the listening socket in SOCKPORT, without waiting for a connection. (The application should then save this returned SOCKPORT value in a separate variable, perhaps called LISTENSOCK, as it may be needed later after SOCKPORT has been again updated.)

Since opening the listening socket is fast (does not require waiting for a client to make a connection request), this eliminates the possibility of the server getting stuck for an indefinite time waiting for a client connection. It can subsequently use the CHECK OP (7) to check if a connection is ready (although it should do so frequently, since the clients do not like to wait very long.) Once a connection is ready to accept (or if it decides it doesn't mind waiting) it can use the ACCEPT OP (1) again, this time with TCPXFLG'ASYNC (and optionally, TCPXFLG'KEEPLISTEN) to accept a connection on the listening socket specified in SOCKPORT. The accepted connection socket is returned in SOCKPORT. (This would overwrite the saved listening socket number, so if you plan to accept further connections on this same listening socket, you must save the listening socket in another variable.)

If the server wants to “stay open for business” while processing the current connection, it should use the TCPXFLG'KEEPLISTEN flag so that the listening socket previously opened (and hopefully saved in a separate variable, i.e. LISTENSOCK) can be left open and reused for the next connection.

As long as the listening socket is left open, any connection requests will queue up, either until the queue fills, or the client gets tired of waiting and cancels the request. The length of the queue has been increased from 5 to 64, which should be more than adequate for most situations.

Note that if you use TCPXFLG'LISTEN, and then don't accept a connection or accept one with the TCPXFLG'KEEPLISTEN flag set, the listening socket remains open, and must be manually closed (in addition to any connection socket) when your program ends. Use the normal OP CLOSE (6) and specify the listening socket (which you hopefully saved in a variable separate from SOCKPORT.)

Opcode 6 (CLOSE), which since TCPX.SBR was introduced in 4.9 has also issued a shutdown command to the socket, now only performs the close operation. A new opcode, 5, has been added to force a shutdown. The distinction between close and shutdown is as follows:

Shutdown sends a signal to the remote end indicating the that sender is shutting down its end of the socket. Since sockets are full duplex, it is possible to shutdown just the sending side, or the receiving side or both. Two new flags, TCPXFLG'SHUTRD (32) and TCPXFLG'SHUTWR (64) have been added to allow the application to take advantage of this capability. (If both, or neither, are specified, the socket is shut down in both directions.) Shutdown does not substitute for close though, and therefore should be followed (eventually) by a normal close.

Close will generally have the same effect as a shutdown in both directions, except in the case where a socket is being shared by a parent and child process. The parent may have accepted the connection, then used XCALL SUBMIT to fork a child, which inherits the connection. Normally the parent would then close its connection socket. In the case, the close will not have any effect on the other end, since the child still has the socket open. (Shutdown, on the other hand, would cause the other end to get an error if tried to read or write to the socket.) When the child closes the socket, since it is the last one at its end to have it open, it will really be closed.

TINKEY

```
xcall TINKEY, char
```

TINKEY.SBR allows you to input a keyboard character if one is available, without waiting if one is not available. The *char* parameter should be a one byte string.

If the session has received the hangup signal and is running in background pending an input operation to terminate, by default, XCALL TINKEY will not be counted as an input operation. If you want it to be counted as an input operation (and thus terminate the session), then specify the –hetcki command line switch when launching A-Shell.

TRIM

```
xcall TRIM, strvar {,flag}
```

TRIM.SBR removes both leading and trailing blanks from the specified string variable. In addition, it can optionally remove other non-printable characters.

Parameters

strvar
is the string variable from which you want to trim the leading and trailing spaces. It does not need a trailing null byte.

flag (numeric)
may be specified as any non-zero value to cause **TRIM** to treat any characters with ASCII values less than 32 (space) or greater than 127 the same as space (i.e. remove them). Like spaces, they must still be prior to the first printable character or after the last one, and prior to the first null, to be removed. Any spaces or other control characters that are embedded between printable characters are left alone.

Comments

The BASICPlus EDIT$() function supports the ability to remove certain types of characters within (rather than just at the beginning or end of) the string. See COMPLP in the *A-Shell Command Reference* for details on EDIT$().

The SBR=TRIMCTL setting in the system parameters file forces **TRIM** to act as if a nonzero flag value had been passed.

TRMCHR

Documentation update January 05; see “Update Notes” below.

```
xcall TRMCHR, status, trmchr'map
```

TRMCHR.SBR provides a means to determine various charactertistics of the terminal running your program. The current number of rows and columns, foreground and background colors, and other information is available.

A-Shell includes both a TRMCHR.BAS sample program, and a TRMCHR.BSI sample include file.

TRMCHR is a direct and exact replacement for the AMOS subroutine of the same name. Refer to the AlphaBASIC XCALL Subroutine User's Manual for complete information.

Update Notes: Build 915 -27 Jan 05

A-Shell/Windows Extension: Under A-Shell/Windows GUI mode, if there is a standard modal dialog box open at the time of the XCALL TRMCHR, WINROW and WINCOL will be set to the usable number of rows and columns in the dialog box.

A-Shell/Server/ATE Extension: Under A-Shell/UNIX (or A-Shell/Windows on a telnet server with ATSD), if the client is ATE, then there is a possible ambiguity in TRMCHR over whether the information returned should be from the server or the client side. In most cases, these will agree, except for the case of WINROW and WINCOL, since dialog box information may only be known to the client. (The server application would have given the command to create the dialog box, but that information, and especially the knowledge of whether the dialog box is still open on the client, isn't necessarily available to TRMCHR.SBR.)

To resolve the ambiguity, if you want the WINROW / WINCOL information to return information about a dialog box displayed on the ATE client, you must set WINROW to -1 prior to the XCALL. In that case, TRMCHR.SBR will query the client for all of the information. Otherwise, it will just use the server's view of the terminal status, which, as of build 915, will mean that the WINROW and WINCOL fields will come back 0. To clarify, the main reason why you would want to set WINROW to -1 before XCALL TRMCHR is if you are interested in knowing the size of the current dialog box on the client display, if any. (It makes no difference under A-Shell/Windows what you set WINROW to, since it always returns the client information in that case.)

UNIQUE

```
xcall UNIQUE, fname
```

UNIQUE.SBR generates a unique filename based on your current jobname. This can be handy for generating print files and other temporary files which are periodically erased.

The paramter *fname* (string variable, 10+ bytes) will be set to <jobnam>.T#@ where # is a digit from zero to nine, and @ is an alphabetic character from A to Z. It scans through these names in order until finding one that is not in use. Thus if the current job is TSKAAA, the first name returned (assuming it is not already in use) will be TSKAAA.T0A, followed by TSKAAA.T0B, etc.

VUESCR

```
xcall VUESCR, textarray, rows, cols, strow, stcol
```

VUESCR.SBR edits a rectangular array of text.

Parameters

textarray
supplies the initial text and receives the updated text. It must be mapped as a string array, with the number of elements greater than or equal to the value of the *rows* parameter, and the size of each element equal to the value of the *cols* parameter.

rows, cols (numeric)
determine the dimensions of the rectangular area of text to be edited, and must correspond to the mapped dimensions of *textarray*.

strow, stcol (numeric)
determine the starting position of the rectangle on the screen. A border will be drawn outside this rectangle, so for best results, the starting row and column should be two or greater.

Comments

VUESCR is actually implemented as a front end to INMEMO, which already supports this exact function (although with different and more complex parameters). It is offered mainly as a convenience to programmers using the Swap (multi-session) utility under AMOS.

WAKNO

```
xcall WAKNO, jobno {,status}
```

WAKNO.SBR (UNIX versions only) wakes up a job that is sleeping (in SLEEP.SBR or SLEEP.LIT).

Parameters

jobno
is the job number of the job to wake up. (See PLYJOB for a way of getting the number of another job.) Note that in order to be able to send a signal like this to another process under UNIX, you either have to be the superuser or have the same effective user id as the target user.

status (optional)
returns the result of the operation:

Value	Description

Value	Description
-1	No such job
0	Success
1	Insufficient privileges to wake target user
3	No such process (job probably aborted uncleanly, leaving a phantom in the job table)

WINFLG

```
xcall WINFLG, flag {,feature}
```

The AMOS routine **WINFLG.SBR** was used within inSight to retrieve information about the current job and terminal environment. *Flag* is a 6-byte floating point variable, and *feature* is any numeric type.

If **WINFLG** was called with just the one argument, it was supposed to return 1 if inSight was supported, else 0. (A-Shell does not support inSight and thus returns 0.) If called with 2 arguments, then it returns *flag* = 1 if the corresponding feature is supported, else 0. The features 1-32 are equivalent to those used by JOBDAT (which see). Features 33-255 correspond to the equivalent TCRT or TAB(-1,x) numbers. Feature 256 indicates Multi or Office (and is thus set to 0 by A-Shell).

Since this routine is used very sparsely in the A-Shell community, A-Shell contains only a minimal implementation. Features 1-32 are supported, as in JOBDAT, but beyond that, A-Shell returns 1 for features 33-149, plus 202 and 203 (screen save/restore). In the case of Windows (except in Telnet mode), it also returns 1 for features 158-162 (mouse functions).

XDEFLT

```
xcall XDEFLT, string
```

string (any length) contains the characters to be inserted into the type-ahead buffer.

These characters will sit in the type-ahead buffer until the next input operation, when they will act like they were typed at that point. If the type-ahead string contains carriage returns, and/or the next input operation is limited in length, then the unused remainder of the type-ahead string will be used on the subsequent input operation, until it is all used up.

XFOLD

```
xcall XFOLD, string, mode, control

xcall XFOLD, string
```

Parameters

string (string of any length)
contains the string to be folded or capitalized.

mode (any numeric type)
may be set to 0 for "word mode" or 1 for "sentence mode." In word mode, each word is capitalized, whereas in sentence mode, typically on the first word of a sentence is capitalized.

control
is a string containing a list of delimiters, followed by a list of exceptions. The format is:

```
/delimiters/exception1/exception2/.../exceptionN/
```

The "/" character in the above string can be any character, as long as it is used consistently and is not part of any of the delimiter or exception strings. The purpose of the exception list is to override the more simplistic capitalization logic, preventing things such as "John Doe Iii" or "Sigmund Freud, M.d." A typical control string might be:

```
/ ,;:).!?/II/III/Jr/Sr/Mr/Mrs/Ph.D./M.D./i.e./NASA/IRS//
```

Comments

If only the first argument is specified, then the string will be capitalized as a name using an implied *control* string of:

```
"/ ,./II/III/Ph.D./M.D.//"
```

This will, for example, convert a STRING passed as "john q. PUBLIC ii" to "John Q. Public II."

XFRMMO

```
xcall XFRMMO, opstat, ch1, link1, ch2, link2
```

XFRMMO.SBR is an utility to move or copy an INMEMO logical memo, either between two memo files or within a single file. If moving between files, it can also change the link size from 2 to 4 or vice versa.

Parameters

opstat (numeric)
On input, must be set to 0 to copy (preserving the source memo) or 1 to move (same as copy followed by deletion of the source memo). On return, it will be set to one of the following values to indicate whether the operation succeeded:

0 = Success
1 = Link error
2 = File full
3 = System error

ch1 (numeric)
is the open file channel for the source memo file.

link1 (B,2 or B,4 to match the format of the source memo file)
is the link to the source memo. Note that if *opcode* is 1 (move), this memo will be deleted (and *link1* returned as 0).

ch2 (numeric)
is the open file channel for the destination memo file (which may be the same as *ch1*).

link2 (B,2 or B,4 to match the format of the destination memo file)
will return the new link to the destination memo.

XLOCK

```
xcall XLOCK, mode, lock1, lock2 {,lock'array | class}
```

XLOCK.SBR provides an abstract symbolic scheme for locking anything, based on a pair of numeric values to which you assign your own meaning.

Parameters

count (B,2)
will return the number of locks set.

mode (B,2) or (B,4)
must be initially set to 3. It will return the number of locks set. Note that the size of this variable determines the format of *array*.

myjob (B,2)
will return your job number.

array (1)
is the first element of an array to receive the list of locks. It should be mapped as:

```
MAP1 ARRAY(###)
   MAP2 LOKJOBNUM,B,2     ! job number owning the lock
   MAP2 LOK1,B,x          ! x=2 or 4, based on size of MODE
   MAP2 LOK2,B,x          ! "      "     "       "
```

Comments

The A-Shell implementation of **XLOCK.SBR** is upward compatible with the AMOS one, so you can refer to the **AMOS.SBR** documentation for more details. The only point of mentioning it here is to note the A-Shell enhancements:

First, A-Shell supports LOCK1 and LOCK2 variables of either B,2 or B,4 (allowing values of up to two billion), whereas the AMOS version is limited to two-byte values.

Second, under A-Shell, you have the option of treating a value previously locked by your job as available (default) or locked. In the latter case, you must set the MIAME.INI parameter SBR=MXLOCK. (See *Configuration* in the *A-Shell Setup Guide* for more details on SBR=MXLOCK, and also on SBR=AXLOCK, which controls whether a certain bug in the AMOS implementation will be emulated or not.)

Third, the optional *class* parameter (B,2) allows you to create multiple locking pools (such as in multiple applications) which do not conflict with each other. The standard version of **XLOCK** uses lock class 0 for normal locks and lock class 256 for pending exclusive locks. Lock class 1 is used by the Aesops routine **LOCK.SBR**, and classes 2 and 3 are used by the OmniLedger routines **XFLOCK.SBR** and **XLOCKS.SBR**, respectively. If you want to define your own, we suggest picking something in the range of 100 to 200.

The fourth enhancement is that of lock "reservations." These were implemented to reduce the difficulty of establishing an exclusive (or wildcard) lock on a file (or other resource, represented by the LOCK1 value) in which multiple jobs are actively placing individual record locks. The problem is that the job desiring the

exclusive lock may have to wait an unreasonable amount of time, since as long as there is one lock in the LOCK1 resource, the exclusive lock cannot be set, and even worse, there is no way to stop additional jobs from coming along and setting locks. The idea of the lock "reservation" is that once set, no new locks can be set on the resource, except by jobs that already have one or more locks set on that resource. This allows those jobs already using the resource to finish their transaction, but any additional jobs will have to wait until after the exclusive lock activity is complete. To use this feature, set MODE to 4, LOCK1 to the resource you want exclusive control of, and LOCK2 to zero. This acts just like MODE 0 (lock but do not wait) except that if the exclusive (wildcard) lock cannot be established, a reservation is set instead (using lock class 256). The calling job gets the same failure code as it would for MODE 0 (i.e. MODE is set to the job number owning the first conflicting lock). However, from that point on, no other jobs can set a lock conflicting with the reservation, unless they already have a lock set in that range. The calling job then repeats the MODE 4 request periodically until the return value indicates success (at which point the reservation will have been converted to a normal lock, or in other words, the lock class will be changed from 256 back to 0).

The fifth enhancement is the addition of two new MODEs which allow you to check for the presence of a lock without actually having to place one. MODE 4 is like MODE 0 (no waiting), and MODE 5 is like MODE 1 (wait until available) except that MODEs 4 and 5 do not actually set a lock. It will be left as an exercise for the clever programmer to figure out what possible motivation there could be for such a capability.

To retrieve a list of locks use this syntax with "Mode" set to 3:

```
xcall XLOCK, mode, myjob, array(1) {,class}
```

XPAINT

```
xcall XPAINT, paintlit
```

XPAINT.SBR was developed to allow screens that were created with AlphaPAINT 2.0 screens to be executed by A-Shell. Under AMOS, these screens are assembled into LIT file executables, which are typically executed via **AMOS.SBR**. Since AMOS machine code will not run on other operating systems directly, A-Shell provides this interpreter that reads the specified LIT file and produces the expected screen display. Note that this only works for LITs generated by AlphaPAINT 2.0.

XPPN

```
xcall XPPN, ppn, device, job, trmdef
```

XPPN.SBR returns the PPN, Device, Job, and Trmdef in the formats shown below.

Parameters

```
MAP1 PPN
   MAP2 Project,B,2
   MAP2 Programmer,B,2
MAP1 Device,S,8             ! (e.g. "DSK1:")
MAP1 Job,S,6
MAP1 Trmdef,S,6
```

Comments

A confusing situation arises when PPN values, which by convention are always represented in octal notation (ranging from zero to 377, rather than zero to 255 as they would if represented in decimal), are stored in numeric variables. Since BASIC has no provision for displaying the value of a numeric variable in anything other than decimal, when the PPN [100,200] is stored in the Project and Programmer fields above and later printed using the PRINT statement, they would display (in decimal) as 64 and 128. To display them in their correct format as octal, you could use the MIAMEX 9: Output hex or octal number.

Another approach would be to set SBR=XPPNOCT in the system configuration file, which causes **XPPN.SBR** to correct for the octal/decimal discrepancy by converting the returned Project and Programmer values such that they display in decimal as if they were octal. Thus, the PPN [100,200] would be returned as Project=100 and Programmer=200, which is probably what you were expecting.

You might wonder why this wasn't done by default. The reason is that under AMOS, PPNs were stored as a pair of bytes (each byte ranging from zero to 377 octal). For closer compatibility, A-Shell internally uses the same representation, even though it is arguably arbitrary since at the operating system level, PPNs are mapped to directories that are purely text strings. However, many existing BASIC programs and subroutines relied on the fact that PPNs could be stored in a pair of single byte variables, and thus at some point we would have had to deal with the conversion anyway. 377 decimal does not fit in a single byte, which is why this particular subroutine uses the somewhat non-AMOS-like technique of storing the PPN as a pair of two-byte variables. For comparison, GETJTB returns the PPN both as a single two byte field and as a pair of three-byte strings (for binary and octal representations).

XTREE

```
xcall XTREE, srow, scol, answer, array, addcnt, coldef, exitcode
     {,erow, ecol, flags, file, mmoclr {,xtrctl, filidx}}
```

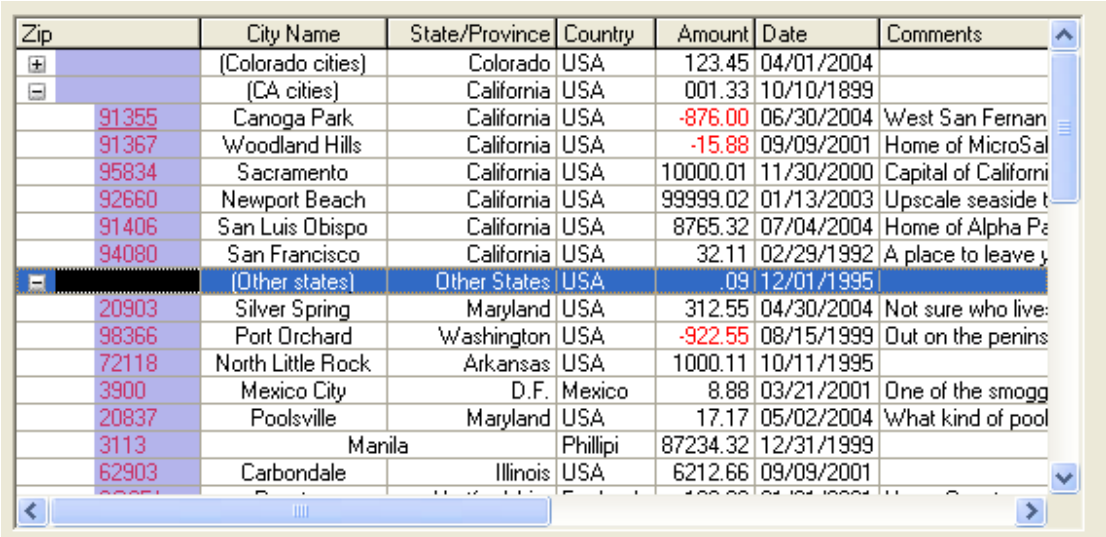
A tree control is similar to a list box, except for the ability to have hierarchical levels. Two examples are shown below.

Introduced in Build 896, XTREE is an upward-compatible variation of PCKLST. While PCKLST ("pick list") was intended for choosing a single from a simple list, XTREE is a full-fledged implementation of a Windows "tree" control, supporting multiple columns and multiple levels of rows. The name "tree" comes from the ability to create a branching hierachy of items, which at each level can be collapsed and expanded. Windows Explorer, for example, is essentially a tree control.

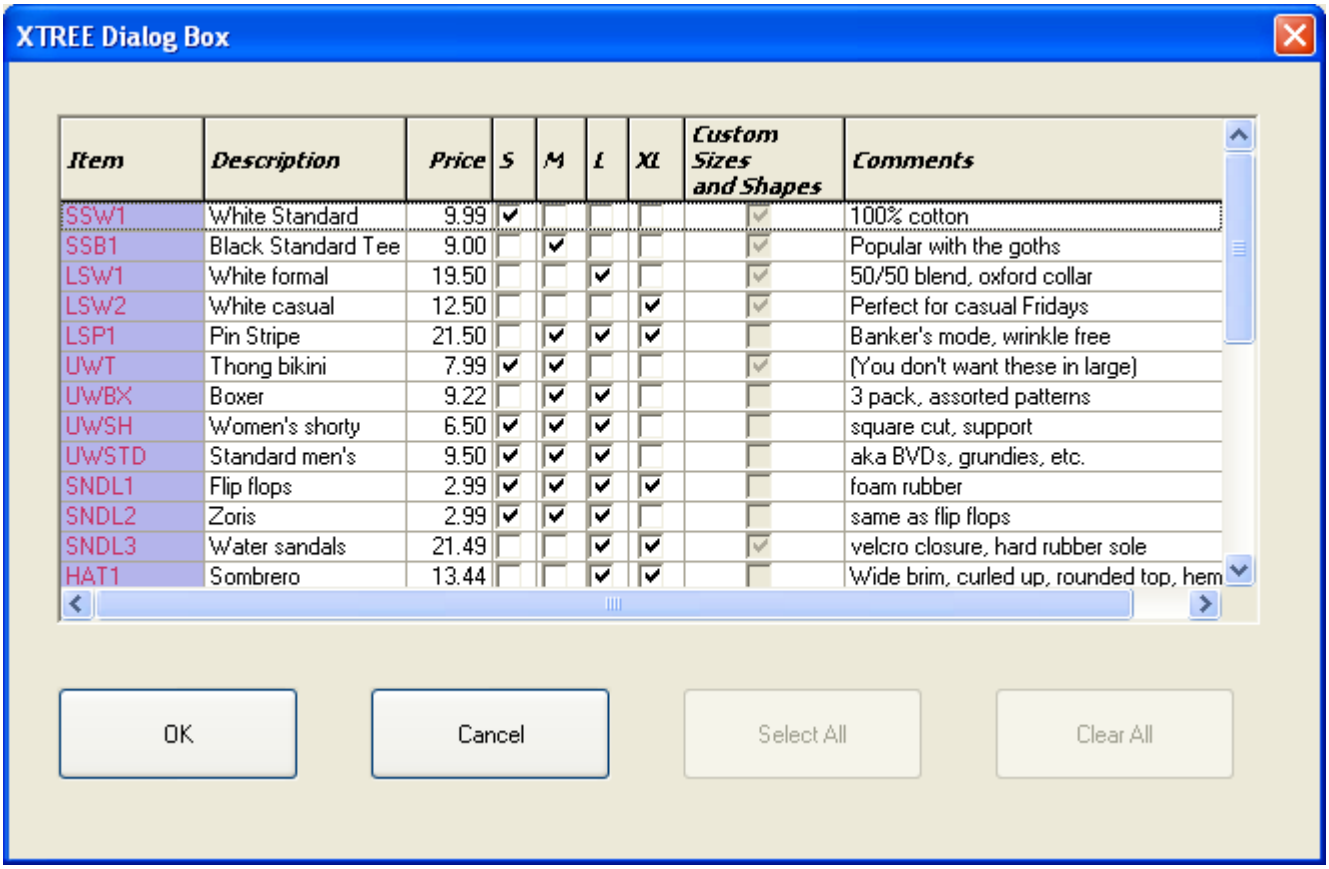
XTREE and PCKLST are actually the same subroutine, so it doesn't really matter what name you use. We split it into two different names simply to make it easier to document and understand, since the uses of XTREE may diverge considerably from those of PCKLST, and also to avoid the potential confusion of people trying to use XTREE features that are not available in a text environment. Our recommendation is for you to use PCKLST if you need text mode compatibility and are (therefore) going to stick with the PCKLST features and semantics (as documented under PCKLST). Otherwise, if you are committed to the GUI environment (either A-Shell/Windows using a driver ending in "G", or with the ATE client) and want to take advantage of the maximum feature set, use XTREE.

Depending on how you call it and the operating environment, the subroutine chooses one of two internal implemenations. For text environments, it uses the INMEMO menu mode to display the list and allow you to choose an item. (This will only be successful though if you limit yourself to the PCKLST feature set and argument semantics.) For GUI environments, it uses a commercial tree control, which we have licensed for A-Shell distribution. It resides in an external module, SftTree_IX86_A_50.DLL, which must be in the same directory where ashw32.exe resides, or in the Windows system32 directory.

Sample tree controls built using XTREE:



Zip	City Name	State/Province	Country	Amount	Date	Comments
⊞	(Colorado cities)	Colorado	USA	123.45	04/01/2004	
⊟	(CA cities)	California	USA	001.33	10/10/1899	
91355	Canoga Park	California	USA	-876.00	06/30/2004	West San Fernan
91367	Woodland Hills	California	USA	-15.88	09/09/2001	Home of MicroSal
95834	Sacramento	California	USA	10000.01	11/30/2000	Capital of Californi
92660	Newport Beach	California	USA	99999.02	01/13/2003	Upscale seaside t
91406	San Luis Obispo	California	USA	8765.32	07/04/2004	Home of Alpha Pa
94080	San Francisco	California	USA	32.11	02/29/1992	A place to leave y
⊟	(Other states)	Other States	USA	.09	12/01/1995	
20903	Silver Spring	Maryland	USA	312.55	04/30/2004	Not sure who lives
98366	Port Orchard	Washington	USA	-922.55	08/15/1999	Out on the penins
72118	North Little Rock	Arkansas	USA	1000.11	10/11/1995	
3900	Mexico City	D.F.	Mexico	8.88	03/21/2001	One of the smogg
20837	Poolsville	Maryland	USA	17.17	05/02/2004	What kind of pool
3113	Manila		Phillipi	87234.32	12/31/1999	
62903	Carbondale	Illinois	USA	6212.66	09/09/2001	


XTREE Dialog Box
Item
Description
Price
S
M
L
XL
Custom Sizes and Shapes
Comments
SSW1 White Standard 9.99 100% cotton
SSB1 Black Standard Tee 9.00 Popular with the goths
LSW1 White formal 19.50 50/50 blend, oxford collar
LSW2 White casual 12.50 Perfect for casual Fridays
LSP1 Pin Stripe 21.50 Banker's mode, wrinkle free
UWT Thong bikini 7.99 (You don't want these in large)
UWBX Boxer 9.22 3 pack, assorted patterns
UWSH Women's shorty 6.50 square cut, support
UWSTD Standard men's 9.50 aka BVDs, grundies, etc.
SNDL1 Flip flops 2.99 foam rubber
SNDL2 Zoris 2.99 same as flip flops
SNDL3 Water sandals 21.49 velcro closure, hard rubber sole
HAT1 Sombrero 13.44 Wide brim, curled up, rounded top, hem
OK
Cancel
Select All
Clear All

Inputs and Outputs

Both XTREE and PCKLST offer a choice of two ways of providing the data for the list:

- Array
- Sequential file

XTREE supports three types of output: (PCKLST only supports the first of these)

- A selected “item” (aka “row” or “record”). This is referred to as “single-selection mode”.
- An array of selected items (“multiple-selection mode”).
- An array of updated cell values (either checkbox values or text strings). This only applies when the data contains editable cells; see *flags XTF_EDITABLE*, and advanced *coldef* column formats E and T below.

Not all features support all types of input or output, or there may be differences in the way the parameters are interpreted or mapped. So as you read the documentation below, please be alert for qualifications which reference one or more input/output combinations, such as “single-selection array mode” or “multiple-select file mode”.

Parameters

Parameters that supply data *to* the routine are indicated with [in], and those used to get data back *from* the routine are labeled [out].

Parameter	Description
srow, scol	[in] Upper left corner of display. See position, below.
erow, ecol	[in] Bottom right corner of display. See position, below.
answer	[in/out] Sets and returns the identity of the selected row(s) (or in the case of checkboxes, the checkbox values).
array	[in] In array mode, specifies an array of strings to be loaded into the listbox.
addcnt	[in] Number of rows to be loaded.
coldef	[in] Defines the column structure.
exitcode	[out] Returns a code indicating how the list was exited; 0 indicates a normal exit via the ENTER key, 1 indicates ESCAPE. See the *flags* parameter for enabling other exit keys. (Same as PCKLST.).
flags	[in] Used to enable additional exit keys and other options.
file	[in] Specifies a file to be used for input instead of the *array* parameter. This is referred to as “file mode”.
mmoclr	[in] Specifies colors for various parts of the pick list.
xtrctl	[in/out] A collection of extended parameters and options.
filidx	[out] Creates an index to the file while it is being loaded into the array.

position

Note that *position* is not a parameter.

The parameters *srow, scol, erow, ecol* mark the upper left and lower right corners of the display box, provided that *flags* contains the option *XTF_XYXY*. Otherwise, they are interpreted as *row*, *col*, *strow*, *endrow* as in PCKLST. Note that since this XTREE supports horizontal scrolling and column resizing, the width of the display box is independent of the width of the data. Also note that the coordinates are relative to the main window or dialog, or to the control specified by XTR'PARENTID.

answer

Answer specifies and returns the identity of the selected row(s). In single-selection array mode, *answer* is treated as in PCKLST, where it may be any numeric type and is interpreted as the selected row number (first row is 1). For multiple selection mode using array or file input (see *flags XTF_MSEL*) it must be an unformatted variable mapped as follows:

```
MAP1 ANSWER
   MAP2 ANSARY(x),S,1     ! each item is "0" if not selected, else "1"
                          ! (x) must be at least as large as addcnt
```

To support both single and multiple selection modes with a single form of the parameter, you can add the following MAP statement for the single selection response:

```
MAP1 ANS,B,4,@ANSWER      ! selected row for single selection
```

You can still pass the unformatted ANSWER variable (rather than ANS), but XTREE will treat it as if a B,4 variable in single selection mode. In other words, in single selection mode, XTREE assumes that an unformatted answer parameter is overlayed by a four-byte binary.

Note that although the rows may be sorted on the screen, either by the user or by XTREE based on the XTR_COLUMNSORT parameter, this does not affect the order of the original data, to which the row numbers returned in *answer* remain relative. In other words, if *answer*=5 on calling XTREE, it will select the fifth row in in the source data (*array* or *file*), which may or may not appear as the fifth row on the screen, and which may be sorted again by the user to yet another position. Similarly, the *answer* value(s) returned will not necessarily reflect the position(s) of the selected item(s) as displayed, rather their position in the source data (which remains unaffected by any sorting performed by XTREE). Also note that if *answer*=0, the default selection will be the first row, except in the case of the append operation, in which case it will be the first row appended. Any other *answer* values are always interpreted as relative to the combined data (original+appended).

When editable checkboxes are present (*flags XTF_CHKBOX*), *answer* must consist of array of N byte items, where N is the number of editable checkbox columns. For example, if there are three checkbox columns, you should map it like one of these:

```
MAP1 ANSWER
   MAP2 ANSARY(x),S,3    ! (x=# rows, 3=# editable checkbox columns)
```

Or, perhaps more clearly:

```
MAP1 ANSWER
   MAP2 ANSARY(x)        ! (x) must be at least as large as addcnt
      MAP3 ANS'CB1,S,1   ! first editable checkbox column
      MAP3 ANS'CB2,S,1   ! second editable checkbox column
      MAP3 ANS'CB3,S,1   ! third editable checkbox column
```

The encoding of these arrays is similar to that for multiple-selection mode, i.e. “0” = unchecked, “1” = checked, and “ ” (blank) causes the cell to be left blank (with no checkbox at all).

When editable text columns are present, the same rule as for checkboxes applies, except that the width of the eliments in the answer array must include the total width of all the editable columns. To continue the example above, if, in addition to the three checkbox columns there were also two editable text columns, one 12 bytes wide and one 25 bytes wide, the correct mapping of the answer array would be:

```
MAP1 ANSWER
   MAP2 ANSARY(x)          ! (x) must be at least as large as addcnt
      MAP3 ANS'CB1,S,1     ! first editable checkbox column
      MAP3 ANS'CB2,S,1     ! second editable checkbox column
      MAP3 ANS'CB3,S,1     ! third editable checkbox column
      MAP3 ANS'TEXT1,S,12  ! first editable text column
      MAP3 ANS'TEXT2,S,25  ! second editable text column
```

Regardless of the relative position of the checkboxes vs. the editable text columns in the source *array*, the checkboxes must precede the text columns in the *answer* array, as shown above.

Unlike editable checkboxes, for which the associated column in the source *array* is ignored, with editable text, the source *array* is used when loading the values for display, but the *answer* array is used for returning the values. Your program can determine which values were changed by comparing the *answer* array to the corresponding columns of the source *array*.

> Individual cells (rows) within an editable text column can be marked to ignore by setting the associated field in the ANSWER arrays to "|" (vertical bar). See RGBignore and program EL.LIT for an example.

Editable text and editable checkboxes are only supported in array mode (not file mode).

array

In array mode, specifies an array of strings to be loaded into the listbox. The interpretation and parsing of these strings into columns is determined by the *coldef* parameter. To pass the array to XTREE, you may either specify a specific starting element (e.g. ARRAY(1)) or an unformatted variable under which the array is defined (ARRAYX in the example below):

```
MAP1 ARRAYX               ! unformatted variable
   MAP2 ARRAY(50),S,100
```

Each array element must have at least one trailing null byte. Note that by using the overlay feature, you could support several array layouts with the same XCALL XTREE statement, i.e.

```
MAP1 ARRAYX2,@ARRAYX      ! overlay on ARRAYX above
   MAP2 ARRAY2(10),S,60   ! array of ten sixty byte strings
MAP1 ARRAYX3,@ARRAYX      ! yet another overlay
   MAP2 ARRAY3(80),S,8    ! array of eighty 8 byte strings
```

In single-selection **file** mode, the *array* parameter is interpreted as an ordinary string to return the text of the selected items (all columns, but with right-justfied columns stripped of trailing blanks.) If the *XTF_FILANS* flag is set, then array is ignored on entry; otherwise it should contain the text of the item to initially select (as returned from the prior call).

In multiple-selection **file** mode, the *array* parameter is ignored. (The selection information is passed via the array of one byte flags in the *answer* parameter, just as in multiple-selection **array** mode.)

addcnt

Addcnt (any numeric type) is the number of items (rows) to be loaded from the data source. Note that in PCKLST, it specified the maximum number of items in the input array, and was ignored for file mode. XTREE obeys it for both array and file mode, but inteprets 0 as meaning to load the entire file.

Note that when loading an array starting from some position other than the first element, *addcnt* must be adjusted so that it doesn't cause the input operation to run past the end of the array.

coldef

This defines the column structure (and headers), replacing the *prompt* parameter in PCKLST. Different syntax schemes are supported, and are referred to as Traditional Syntax**,** Simple Multi-Column Syntax**,** and Advanced Syntax.

Traditional Syntax

The traditional syntax applies to single-column lists and consists of an arbitrary string to be used as a title. Unlike the PCKLST prompt parameter, XTREE ignores any bottom prompt (which was specified in PCKLST by inserting a chr(13) in the string to separate the top and bottom portions.) Note that there is one important limitation on the contents of the string: it must not contain any groups of two or more spaces enclosed by non-spaces, since this is how XTREE distinguishes between the traditional and simple multi-column syntax.

Simple Multi-Column Syntax

The simple multi-column syntax allows simple multi-column lists to be easily defined for GUI mode, in a way that is backwards compatible with text mode. It consists of a simple string containing the column titles spaced to match the actual column spacing in array.

> If the *XTF_FCOLDEF* bit is set in the *flags* parameter, then the coldef parameter is ignored and the first line of the input data is used in its place. (This is particularly convenient for simple multi-column file based data, since it allows you to keep the column titles with the data itself.)

In order to recognize that a word is the beginning of a new column and not a continuation of the previous column, it must be preceded by two or more spaces. For example:

```
MAP1 COLDEF,S,100,"Zip   City           State"
!                  12345678901234567890123456"
```

The above code would define a three-column list. In order for this to work, the layout of the input data (either in *array* or the *file*) must match this layout. Note that one limitation of this method of defining the columns is that the actual column width in characters must be at least two columns wider than the column title. For example, if a column is to contain just a Y or N, but you want to display the title “Back Ordered?”, then you would have to pad the column with about fifteen spaces to allow the column data to match up with the title string (unless this was the last column.) The **advanced** syntax allows you to transcend this limitation.

The columnar interpretation of *coldef* presents a possible compatibility issue for existing applications counting on XTREE being upward compatible with PCKLST, namely that any *prompt* string containing alphanumeric (non-space) characters separated by two or more spaces would be interpreted as a multi-column definition in the new version. The chance of this happening, though, seemed remote enough (and in the case of text mode, harmless enough), that we decided it was reasonable to simply require you to remove the ambiguity in any existing *prompt* strings by eliminating the contiguous blanks if you are interested in using the GUI mode.

Advanced Syntax

The advanced syntax allows for the specification of column attributes as well as widths. To use it, you must set bit *XTF_COLDFX* in the *flags* parameter, in which case *coldef* is interpreted using the following syntax:

```
cpos~ cwidth~ ctitle~ cformat {~option1= value1{ ~option2=
     value2...} ~~cpos~ cwidth...~~
```

Note that an Advanced Syntax Example is provided after the following notes.

The above syntax specification consists of a series of column definitions, one per column, delimited by a pair of tildes (~~). Each column definition begins with four mandatory parameters (*cpos*, c*width*, c*title* and c*format*) followed by zero or more optional parameters using the form (*optionN*=*valueN*). (This is admittedly cryptic, but has the virtues of being flexible, easily extensible, compact, and easy to map.)

The *cpos* and *cwidth* parameters define the starting position and width of the column relative to the input source (whether array or file). The first position is 1. If you are defining the columns consecutively (in the same order as the input, then you can specify 0 for *cpos* in which case it will assume that the column starts immediately after the end of the previous column. For comma-delimited file input, set *cwidth* to 0 and *cpos* to the field number.

> You may set both *cpos* and *cwidth* to 0 to define a pseudo-column, which doesn't not count in the column numbering scheme, nor take up any space in the control, but may instead be used to define advanced options that are not column specific or which may need to be defined in order to be referenced later. See the column options PopupMenu, HdrFont, HdrScale, Font and Scale below.

The *ctitle* parameter may be a text string of any length to display as the column title. Note that the initial column display width is determined based on the largest string in either the title or any of the data items for that column. If you don't need a title for a particular column, set it to a blank (“ ”), otherwise you'll end up with two adjacent ~ delimiters which will confuse the parser. You can embed line breaks in the header for a column using CR+LF (i.e. chr(13)+chr(10)).

> You may embed a tilde in the *ctitle* string (so as to appear in the column heading when the control is displayed) by replacing it with the HTML-inspired encoding "**%7e.**"

The *cformat* parameter is made up of one or more of the following characters, which are used to determine how the column should be justified and sorted. Note that they do not cause the data in the column to be re-formatted. For example, if a column contains dates, it is up to the calling program to preformat them appropriately to match the *cformat* specification. (The only purpose of defining such a column as a date is to allow sorting to work properly.) The most generic and common type would be S (string).

The advanced *coldef* parameter must be terminated with a pair of tildes (~~).

Cformat	Meaning
@	For multi-level lists, a column must be defined using this type, and containing a digit indicating the hierarchical level of each row ("0" thru "9" with 0 being the top level.) Note that you would almost always want to also use the H code to hide this column; otherwise you'll have a visible column containing the level indicators. Due to some internal complexities in the control, it may not be possible to hide the first column; to avoid this problem, make sure this @ column is not the first one defined.
#	Numeric data
C	(Upper case "C") Color definition column, allows you to apply custom colors to specific rows. Column must contain a palette color number (0-15) to be applied to the foreground color of the remaining columns in that row. Leave column blank to use the default color scheme for that row. Note that you would almost always want to also use the H code with this column type to hide it.
c	(Lower case "c") Identical to "C", except that it takes effect starting from the first visible column even if the color definition column is not the first column. This is a workaround for a limitation in which trying to make the first defined column invisible doesn't always work. (XTREE seems determined to make the first column reappear.)
D	Date in *mm/dd/yr* or *dd/mm/yr* format. The current language definition will be used to determine whether the data is in *mm/dd/yr* or *dd/mm/yr* format. *yr* may be in YY or CCYY format.
d	Date in dd-mon-yr format.
E	Column contains editable text. A single click on the cell will open up an edit window, allowing the text to be changed. This is a primitive edit operation, with no validation. The edited text is returned separately from the *array* in the *answer* parameter (which see).
f	Font definition column. Same concept as the color definition column (cformat c) Column should be 1 character wide, and contain a space or digit 0-4. Space or 0 causes the default font to be used for that row, else references the first thru fourth special fonts defined previously using the Font and/or Scale options with a pseudo column.
H	Hidden column. Allows you to start with a fixed format source of data (either array or file) and customize it at runtime but making selected columns invisible. (Hidden columns are loaded into the listbox just like any other data, so you wouldn't want to over-use this technique with large data sets due to the wasted overhead.)
M	Items in column may contain embedded CRLFs causing it to display using multiple lines. Note that *XTR_ITEMLINES* determines the maximum number of lines of text displayed in a cell. (Cells that exceed this number of lines will display an elipsis or "+" to indicate that there is more data than can be displayed.) Also see *flags XTF_VARY* (variable item height). Note that you cannot use embedded CRLF line breaks in file mode, because CRLF is used for the record terminator.
K	Keep column position (i.e. don't allow it to be reordered by the user). Needed (as an override) only when *flags XTF_REORD* set
L	Lock column (don't allow it to be resized by the user)
O	(capital Oh) Indicates that this column is allowed to overflow into the next column (provided the next column contains the "o" option and the cell is empty.)

Cformat	Meaning
o	(lower case oh) Indicates that an overflowing previous column may overwrite this column, provided the cell is empty. Note this setting is not just automatic based on the prior column, because of the possibility of the user reordering the columns at runtime.
S	String data (left justified by default)
T	Editable checkbox. You must also specify the *flag XTF_CHKBOX* when using checkboxes. Note that although the checkbox column must correspond to a column in the input data (typically a one-character column), the contents of that column in the input data are ignored, and instead, the checkbox settings are retrieved and returned via the *array* parameter.
t	(lower case) Non-editable checkbox. This should be a single character column, whose contents are "0" for an unchecked box, "1" for a checked box, or blank for a blank cell (with no checkbox). Non-editable checkboxes act just like other data columns, and do not require the *XTF_CHKBOX flag* or have any correspondence with the *array* parameter.
W	Auto-wrap columnar data onto multiple lines. (See note about *XTR_ITEMLINES* above.)
X	For use with editable text (E) or checkbox (T) columns; forces an exit for validation with EXITCODE -48 immediately after each cell edited. See Editable tree controls
<	Left justify column (default)
|	Center justify column
>	Right justify column. Note that in file mode, this causes the trailing blanks for this column to be stripped, which affects the return data. See the *flags* option *XTF_FILANS*.

Zero or more of the following options may be specified to further define column options. Remember that each option must be specified using the syntax option=value and delimited on either side by a tilde. (Both *options* and *values* are case sensitive.)

Color

A pair of digits, comma separated, representing the desired foreground and background text color for the column, from the A-Shell palette. Use only if you want the column to have a different color scheme from the others. E.g.: Color=1,2. (Foreground color #1, background color #2).

Dspmin

Similar to **Dspwid** except sets the minimum display width. (Unless the column is locked with the cformat L option, the user can resize the column, subject to this minimum.) (e.g. Dspmin=5)

Dspwid

Sets the initial display width of the column (in character positions based on the standard character grid). For example, if set to 10, the column will occupy the same width as ten ordinary fixed-pitch characters (although it may display more in the typical proportional font.) If not specified, the initial column widths will be set

wide enough to accommodate the longest data. This option is mainly useful for columns that may contain variable length text (see *cformat* **W** option above), lest they expand to fill the entire pick list width, forcing the user to have to scroll horizontally (or manually resize the column) to see the other columns. (e.g. **Dspwid=10**)

Font

A font face name and/or font attribute code, separated by a comma. (e.g. Font=Lucida Console,1537) Either part can be omitted to accept the default for that parameter, in which case the comma should also be omitted. Font attribute codes consist of a sum of the following values. (There may not be a meaningful difference between some of the stroke weights – these are just hints to the font mapper.)

Code	Meaning
0	Normal
1	Italics
2	Underline
4	Strikeout
256	Thin
512	Extra light
768	Light
1024	Normal
1280	Medium
1536	Semi bold
1792	Bold
2048	Extra bold
2304	Heavy

When associated with a real column, the font defined affects just that column. When associated with a pseudo column (i.e. with *cwidth* = 0) it just serves to define the font so that it can be referred to by data in a font definition column for applying specific fonts to individual rows (see *cformat* **f** in the table above). Also see Scale.

HdrFont

Same format and concept as **Font** (above), except that it affects the font of the header row, rather than the data items. By convention, you would define the HdrFont as part of a pseudo column (i.e. a column whose *cwidth* = 0) but, regardless of which column this command appears in, it affects the entire header. Currently there is no way to have different fonts in different column headers. If more than one column specifies a **Hdrfont**, only the last one will have an effect.

HdrScale

This is to **Scale** as **HdrFont** is to **Font**.

PopupMenu

Defines a popup menu (aka "context menu") that will appear when the user right-clicks anywhere on a row. The PopupMenu definition consists of a series of pairs of text/cmd items with the following syntax:

```
PopupMenu=text1,cmd1;text2,cmd2;...;textN,cmdN
```

The <text> fields define what appears on that line of the popup menu, while the <cmd> fields define the key sequence that is transmitted if the user clicks on the option. See Virtual Key Symbolic Names.

To create a horizontal separator bar, set the <text> field to a string of dashes and leave the <cmd> field null (with no spaces between the dashes, the comma, and the semicolon.).

For example, consider this PopupMenu definition:

```
COLDEF = COLDEF "0~0~X~S~ PopupMenu=Edit, VK_xF101; Apply, VK_xF102; " "-----,;
   Reverse, VK_xF103~~"
```

This would create a popup menu, which, when the XTREE control was right-clicked, would display three choices (Edit, Apply, and Reverse) with a horizontal separator between the last two items. If the user then clicked on one of the items in the popup menu, XTREE would exit, setting exitcode to -101, -102, or -103, depending on the item. The effect would be exactly as if you had defined buttons which transmitted the same key codes.

In addition to the exitcode, the XTR'XROW and XTR'XCOL fields in the xtrctl parameter will be set to indicate the logical row and column which where the right-click occurred.

> In the Advanced Syntax Example, below, although the column title (X) and type (S) are ignored, they must be non-null since two consecutive tildes mark the end of the column definition. Also, the column position and size parameters are both set to zero, since the popup menu is not related to a particular column. See the note under Advanced Syntax for further comment on using pseudo-column 0 for non-column specific features.

RGBbg

Same as RGBfg but for defining the background color of a column. (e.g.RGBbg=192,161,133)

RGBfg

A set of three comma-separated numbers representing the RGB values for the desired foreground (text) color of the column. (This is an alternate way to define a custom color indepedently from the A-Shell palette.) (E.g.: RGBfg=100,200,50) Note that RGB values each range from 0 to 255. You can experiment with them on the A-Shell Settings..Colors dialog (click on any color then click on define custom colors.)

RGBignore

Applies only to editable text columns, and is equivalent to RGBbg except that it only applies to cells that are marked to ignore. (See answer.)

Scale

A font scaling percentage (100=normal). (e.g. Scale=50)

When associated with a real column, the font defined (indirectly via the scale factor) affects just that column. When associated with a pseudo column (i.e. with *cwidth* = 0) it just serves to define the font so that it can be referred to by data in a font definition column for applying specific fonts to individual rows (see *cformat* **f** in the table above). Also seeFont.

Advanced Syntax Example

To clarify the advanced form of the *coldef* syntax, consider this example:

```
COLDEF="0~0~x~H~Font=Playbill~~"                          ! row font #1
COLDEF=COLDEF+"0~0~x~H~Font=,2048~Scale=200~~"            ! row font #2
COLDEF=COLDEF+"0~0~x~H~HdrFont=Times Roman~~"             ! Header font
COLDEF=COLDEF+"0~0~x~H~HdrScale=150~~"                    ! Header scale
COLDEF=COLDEF+"1~1~x~fH~~"                                ! font column
COLDEF=COLDEF+"2~1~x~cH~~"                                ! color column
COLDEF=COLDEF+"3~15~PO #~SK~~"                            ! col 1
COLDEF=COLDEF+"18~10~Order"+chr(13)+chr(10)+"Date~D|~~"   ! col 2
COLDEF=COLDEF+"29~12~Amount~#~Dspmin=6~~"                 ! col 3
COLDEF=COLDEF+"52~40~Notes~SMW3~RGBfg=240,20,50~~         ! col 4"
```

This defines four pseudo columns (for the purposes of defining two row fonts, a header font style and a header font scale), plus two special controls columns (font and color) which do not display but which do contain data, and four visible data columns.

Of the visible data columns, the first is supplied by the character positions 3 thru 17 of the data source records (*array* or *file*), has a title of “PO #”, contains string data, (left justfied by default) and may not be reordered. The second occupies positions 18-27 of the source data record, has a title of “Order Date” (broken into two lines), contains dates using the format mm/dd/yr (or dd/mm/yr, depending on the language definition), and is center justified. The third occupies positions 29-40 of the source records, has a title of “Amount”, contains numeric data (right justified by default), and has a minimum width of 6 standard character cells. The last column comes from positions 52-91 of the source records, has a title of “Notes”, contains string data, supports both explicit line breaks (embedded CRLF) and also auto-wrap based.. It also specifies a custom text color RGB(240,20,50). Note that the last column width will always be set to accommodate the longest cell data, so the auto wrap option won't have any effect, unless the column is moved by the user (via drag and drop on the column header), in which case it can be resized and the wrap will have a significant effect.

Regarding the last column, note that because its definition starts in position 52 of the input data, while the previous column ended in position 40, the input data in positions 41-51 will be ignored (not loaded into the control, even in a hidden column). The display width setting of 20 refers to standard character cells (which are typically larger than the average proportional-font character width) so the column may display more or less than twenty characters for reach record. There is probably not much point in limiting it to twenty in this

example, since it is the last column anyway. The only effect of the setting would be to limit how much screen space the column takes up when the control is scrolled all the way to the right. See the *xtrctl XTR_ITEMLINES* and *XTR_TRUNCATED* parameters and *flags XTF_VARY,* which affect how the column will deal with contents that doesn't fit in the display area provided.

flags

This parameter may be used to enable additional exit keys and other options. Specify as the sum of zero or more of the following. The symbol definitions come from **Error! Reference source not found.**.

Symbol	Description
XTF_FKEY	Allow F1-F16 (returning Exitcodes -1 thru –16)
XTF_LEFT	Enable Left arrow (Exitcode -40)
XTF_RIGHT	Enable Right arrow (Exitcode –41)
XTF_UP	Enable Up arrow (Exitcode -42)
XTF_TAB	Enable TAB key (Exitcode -44), and Shift-TAB (Exitcode -35)
XTF_HOME	Enable HOME key (Exitcode –45)
XTF_END	Enable END key (Exitcode –46)
XTF_NOAS	Disable auto-shrink of box height to match amount of data. (This flags is ignored by XTREE since it doesn't support auto-shrink.)
XTF_MODELESS	Leave listbox on the screen after exit from the subroutine.
XTF_TIMOUT	200 second timeout. (Acts as if user had hit ESC on timeout.) [Not yet implemented in XTREE]
XTF_FST	Fast mode. See original version for text mode. In XTREE GUI mode, causes a select operation to exit on a single click (rather than a doubleclick or other exit key).
XTF_FCOLDEF	For file mode, first record is interpreted the same as the **simple multi-column** *coldef* syntax.
XTF_XYXY	Alternate coordinate mode.
XTF_SORT	Allow sorting.
XTF_REORD	Allow column reordering by drag and drop. (Note that if enabled globally, it can still be turned off for individual columns.)
XTF_MSEL	Allow multiple selections.
XTF_MLVL	Multi-level (hierarchical) row support. This will be set automatically if the first column defined in *coldef* uses the @ code.
XTF_COLDFX	*Coldef* parameter uses the **advanced** syntax.
XTF_VARY	Variable height rows. Calculation is subject to maximum number of text lines per item as set in the *xtrctl XTR_ITEMLINES* parameter.
XTF_NOSEL	Exit immediately, rather than waiting for the user to make a selection. (This only makes sense in conjunction with *XTF_MODELESS*; also see *xtrctl XTR_OPCODE*)
XTF_DISABLE	Disable the tree control on exit.
XTF_DEL	Enable DEL exit (Exitcode –47)

Symbol	Description
XTF_FILANS	In single-select file mode, return result via row number in *answer* parameter.
XTF_CTRLC	Causes Control-C to set Exitcode 10 (as in INFLD) instead of 1.
XTF_EDITABLE	Declares that the tree contains editable text (i.e. one or more columns with column format T or E). Without this flag, edits may appear to work, but the interpretation of the *array* parameter will probably be wrong, as it must be interpreted in a scope where the *coldef* parameter is not available.
XTF_TOTALS	Declare that last line of data contains totals (and thus is not to be sorted).
XTF_NO-REDRAW	When used with XTR'OPCODE 4 (select from existing control), defeats all redraw/refresh of data, which makes for a "cleaner" re-entry, assuming that none of the parameters which might affect the display have changed. Without this flag, re-selecting from an existing control will reset the columns, possibly adjust the vertical scroll, etc. The sample program XTRA3 demonstrates this.

XTF_MODELESS

Leave listbox on the screen after exit from the subroutine. In GUI mode, this also leaves the control defined, for possible re-entry or refresh later. (See *XTF_NOSEL* and *xtrctl XTR_OPCODE*.) (The symbol name, *XTF_MODELESS*, is meant to represent “modeless”, since this makes the pick list act something like a modeless dialog, i.e. a dialog that allows you to click back on the main window without first closing it, as opposed to the normal “modal” dialog.) Also see *XTF_DISABLE*.

XTF_FCOLDEF

For file mode, first record is interpreted the same as the **simple multi-column** *coldef* syntax. (This is makes it easier to set up simple multi-column lists, eliminating the need to make the *coldef* spacing match the spacing of the columns in the file records.)

XTF_XYXY

Alternate coordinate mode (required in GUI mode to allow precise specification of the listbox width and position). Indicates that the first two parameters are to be interpreted as the upper left corner row/col, and the 8th and 9th parameters as the lower right corner row/col. This flag should probably ALWAYS be set.

XTF_SORT

Allow sorting (by clicking on a column header). Note that sorting only affects the display; it does not affect index of the selected items returned in array mode. (In other words, if sorting causes the first row of the array to be sorted to the end, and it is selected, *answer* will still be returned as 1.)

XTF_MSEL

Allow multiple selections. The operator accomplishes this by holding down the CTRL key while clicking to make a selection (or SHIFT while clicking to select a range). In this mode, the *answer* parameter must be mapped as an array of one byte strings, which will be returned representing the status of each row ("1"=selected).

XTF_DISABLE

Disable the tree control on exit. Only makes sense if *XTF_MODELESS* also set. This might be useful to either visually make it clear to the user that the control is not active, or to eliminate the need to process a keyboard click string for when the user clicks on the control outside of the confines of XCALL XTREE. See *XTR_KBDSTR*.

XTF_FILANS

In single-select file mode, return result (and select default) via row number in *answer* parameter, rather than row text in *array* parameter. Selected row data is still returned in the *array* parameter though. Identifying the selection via the row number rather than the row data eliminates ambiguities when portions of the row data are common to more than one row, as well as when columns using the right justification format option, but not right-justified in the file data, get stripped.

file

This specifies a sequential file to be used for input instead of the *array* parameter. When non-null, the interpretation of *array*, *addcnt*, and *answer* parameters changes as follows.

If *neither* the multiple selection (*XTF_MSEL*) nor *XTF_FILANS* flags are set, then *array* becomes an ordinary string, which is used to pass in the default selection (text) and to get back the item chosen by the user and *answer* is ignored.

If *XTF_MSEL* is *not* set but *XTF_FILANS* is set, then *array* is still treated as an ordinary string which returns the text of the item chosen, but *answer* is used exactly as in array mode (to return the row number of the selected item, and to pass in the default row number).

If the multiple selection flag (*XTF_MSEL*) is set, then *array* is ignored and *answer* is treated as an array of one byte selection flags, exactly as for multiple-selection array mode. *addcnt* should be set to 0 to load the entire file, or else set to the number of records (lines) to be loaded.

The default selection process (in single-selection mode) works only on the first column. If the cells in the first column are not all unique, then you may not end up with the desired item selected. The only workaround is to add a first column that is unique, or to switch to the array mode, which identifies selections by the array index. Also note that XTREE supports the PCKLST "hidden text" feature, which allows you to pass hidden text in the last column of the data by preceding it with a \. Although this technique is useful for associating control information with list items without having to display it, a better way to do this with XTREE is to just define a hidden column. The \hidden text feature is preserved just for compatibility with PCKLST.

mmoclr

As in PCKLST, may be used to specify colors for various parts of the pick list, although in the GUI mode, the intepretation of color –1 is slightly different. Instead of specifying the current foreground or background color, it specifies the Windows default color for that particular part of the pick list display. (This will result in a typical-looking listbox with black text on a white background, while the column headers are gray buttons.) The format of MMOCLR is:

```
MAP1 MMOCLR
    MAP2 BFCLR,B,1,-1     ! Border Foreground (NA in GUI mode)
    MAP2 BBCLR,B,1,-1     ! Border Background (NA in GUI mode)
    MAP2 TFCLR,B,1,-1     ! Text Foreground (default black)
    MAP2 TBCLR,B,1,-1     ! Text Background (default white)
    MAP2 AFCLR,B,1,-1     ! Arrow Foreground (NA in GUI mode)
    MAP2 ABCLR,B,1,-1     ! Arrow Background (NA in GUI mode)
    MAP2 PFCLR,B,1,-1     ! Title Foreground (default black)
    MAP2 PBCLR,B,1,-1     ! Title Background (default gray)
    MAP2 WFCLR,B,1,-1     ! Warnings & messages Foreground (NA in GUI)
    MAP2 WBCLR,B,1,-1     ! Warnings & messages Background    "  "
    MAP2 SFCLR,B,1,-1     ! Orig. Status line Foreground      "  "
    MAP2 SBCLR,B,1,-1     ! Orig. Status line Background      "  "
    MAP2 RFCLR,B,1,-1     ! Reserved (unused) Foreground      "  "
    MAP2 RBCLR,B,1,-1     ! Reserved (unused) Background      "  "
```

The individual fields within MMOCLR are shown above initialized to -1, which is what you must do to get default colors, since color 0 is black. (Specifying MMOCLR with all the fields un-initialized will result in a not-very-interesting black on black display.) Note that the colors for individual columns can be overridden using the **advanced** form of the *coldef* parameter.

xtrctl

This is a collection of extended parameters and options. If omitted, suitable defaults will be supplied. For those parameters which are essentially Boolean options, 0=false and 1=true. Most of the fields in **xtrctl** are input only, but a few, such as `XTR_COLUMNACTIVE`, are updated as well; these are indicated with "[in/out]" in the table below. It is highly recommended that you include in your programs **Error! Reference source not found.** to define the xtrctl map structure, and **xtree.def** to define the symbols.

```
MAP1 XTRCTL
    MAP2 XTR'OPCODE,B,1           ! [in] Opcode. See XTR_OPCODE below.
    MAP2 XTR'CTLNO,B,1            ! [in/out] Specifies control to use. See XTR_CTLNO.
    MAP2 XTR'ITEMLINES,B,1        ! [in] Max number lines of text displaying in a cell
                                  !(if flags XTF_VARY not set, affects all rows)
    MAP2 XTR'TREELINESTYLE,B,1    ! Style of lines linking child lines to parent:
                                  ! 0)none, 1)solid, 2)dotted
    MAP2 XTR'SHOWBUTTONS0,B,1     ! [in] Show level 0 tree btns (0=no, 1=yes)
    MAP2 XTR'SHOWBUTTONS,B,1      ! [in] Show expand/collapse btns for lvl 0 (These
                                  ! "buttons" are in addition to the standard
                                  ! clickable +/- indicators)
    MAP2 XTR'SHOWGRID,B,1         ! [in] Show grid lines? (0=no, 1=yes)
    MAP2 XTR'GRIDSTYLE,B,1        ! [in] 0=vert solid, 1=horz solid, 2=both, 3=vert
                                  ! dotted, 4=horz dotted, 5=both
    MAP2 XTR'TRUNCATED,B,1        ! [in] Show dots if text truncated?
    MAP2 XTR'SELECTAREA,B,1       ! [in] Clickable area and style. See XTR_SELECTAREA.
    MAP2 XTR'FLYBY,B,1            ! [in] Fly-by highlighting? (0=no)
    MAP2 XTR'SCROLLTIPS,B,1       ! [in] Show scroll tips (0=no)
    MAP2 XTR'COLUMNACTIVE,B,1     ! [in/out] Col # with focus (for kbd search)
```

```
MAP2 XTR'COLUMNSORT(3),B,1     ! [in/out] first, second, third sort columns
MAP2 XTR'SORTORDER(3),B,1      ! [in/out] 0=ascending, 1=descending
MAP2 XTR'KBDSTR,S,10           ! [in] Click code (see below)
MAP2 XTR'USETHEMES,B,1         ! [in] Use XP themes? (0=no)
MAP2 XTR'PARENTID,B,2          ! [in] ID of parent of ctl (e.g. a dialog)
MAP2 XTR'SHOW3D,B,1            ! [in] Use 3D style? (0=no) See sample screen below.
MAP2 XTR'HIDEHEADER,B,1        ! [in] Hide headers? (0=no)
MAP2 XTR'XNAVCOD,B,1           ! [in/out] Used in cell editing (see below)
MAP2 XTR'XNAVMASK,B,1          ! [in/out] "    "     "    "
MAP2 XTR'XROW,B,4              ! [in/out] Row of cell being validated (see below)
MAP2 XTR'XCOL,B,1              ! [in/out] Col of cell being validated
MAP2 XTR'EXPANDLEVEL,B,1       ! [in] 0=collapse, #=expand thru level # (9 max)
MAP2 XTR'SKEY,S,10             ! [in] search key. See
                              XTR_SKEY
MAP2 XTR'UNUSED,S,18           ! for expansion
```

XTR_OPCODE

XTR_OPCODE determines the operational mode, from one of the following choices. (See **Error! Reference source not found.** for the symbol definitions and see the sections below the table for more description of each opcode.)

Symbol	Description
XTROP_CREATE	(0) Normal usage. Create a new control.
XTROP_REPLACE	(1) Use existing control but replace data within it
XTROP_APPEND	(2) Append data to an existing control
XTROP_DELETE	(3) Delete an xtree control
XTROP_RESELECT	(4) Reselect from an existing control
XTROP_DELSEL	(5) Delete selected rows

XTROP_CREATE

In this, the "normal" usage mode, a single XCALL XTREE causes the control object to be created and loaded with data. Unless the *XTF_NOSEL* flag is specified, the control will then receive the focus and wait for the user to perform an action that causes it to exit. On exit, unless the XTF_MODELESS flag is set, the control is deleted. You may specify the control number (0-9) in XTR_CTLNO or set it to -1, in which case the next available XTREE number will be used.

XTROP_REPLACE

The replace operation allows you to replace the existing items in the control with the new set. This is similar to the normal operation, except that the control does not have to be recreated. If *flags XTF_NOSEL* is not set, the control gets the focus and waits for a user selection (as in normal mode). Otherwise, it exits immediately (in which case *flags XTF_MODELESS* must be set for the operation to make sense). With both *XTF_MODELESS* and *XTF_NOSEL* set, opcodes 1 and 2 allow you to use the control as a sort of display holding area (perhaps like a shopping cart), allowing items to be added to it for temporary display purposes (and presumably later allowing the user to select items to be cancelled, changed, finalized, etc.)

XTROP_APPEND

The append operation is just like replace, except that the data passed in the source array or file is appended to the existing data in the control. In the case of array mode, the best way to handle this is to define an array large enough for all the items. For example, you might plan on a maximum of 1000 items, but initially load only 50. If you then append another 75, put them in the array positions from 51 to 125, and specify ARRAY(51) as the array parameter and 75 for the addcnt parameter. This way, your array continues to correspond to the current state othe list, so that you can make sense of the returned selection information.

XTROP_DELETE

The operation deletes the control, and is only necessary when the control was created with the XTF_MODELESS flag (otherwise it is deleted automatically). Note that you can also delete the XTREE control (like any other control) by clearing the screen with TAB(-1,0);

XTROP_RESELECT

Reselect is similar to append, except without adding any new data. It is used to "re-enter" an existing modeless control without the overhead of recreating the control or reloading the data. The initial selection(s) is/are determined by the answer parameter.

If the control contains editable cells, then the *XTR'XROW*, *XTR'XCOL*, *XTR'XNAVCOD* and *XTR'XNAVMASK* parameters in xtrctl determine the position of re-entry.

XTROP_DELSEL

This opcode deletes just the row(s) identified by the answer parameter. It acts similarly to *XTROP_RESELECT*, not loading any new records. But instead of selecting the row(s) indentified by *answer*, it deletes them. It then waits for a new selection, unless the *XTF_NOSEL* flag is set, in which case it exits immediately. Note that the original data rows are not internally renumbered, so if you are trying to keep your internal array synchronized with the control, you should only mark the items deleted but not compact the array. For example, if you delete item #2, then reselect from the same array, item #3 will appear it position #2 on the screen, but if you select it, it will still be identified as item #3.

XTROP_DELSEL will mainly be of use in situations where you have a large number of items, making it inefficient to delete and reload the entire control each time you want to get rid of one. This would be particularly true if you didn't really need to keep track of the items being deleted within the context of the original array. (For example, you might start with a list of possible items to choose from, and then as you choose them, they are deleted from that list and moved to another list, like in a shopping cart application. You don't really need to keep track of the state of the original list, because when the order is confirmed, you would work from the list of chosen items, rather than from the list of available items.)

See the sample program XTRA4 for an illustration of moving items between two XTREE controls.

XTR_CTLNO

When multiple XTREE controls are in simultaneous use, specifies which one (range is 0 thru 9) to use for this call. If *XTR_OPCODE = XTROP_CREATE*, you can set *XTR'CTLNO* to –1 to have XTREE automatically assign the next available control number. (If the control is modeless, you'll need to save the returned *XTR'CTLNO* value for referencing it in subsequent calls.)

XTR_SELECTAREA

This field actually encodes two separate parameters for related sets of options. One defines the area in which you may click to make a selection, and the other defines how the selection is highlighted. Choose one from the Area options and one from the Style options:

Area	Description
XTRSEL_AREA_ANY	(0) anywhere on row, including tree lines, row headers
XTRSEL_AREA_CELLS	(1) any cell of row
XTRSEL_AREA_CELL1	(2) only first cell of row
XTRSEL_AREA_CELL1T	(3) only text in first cell of row
Style	
XTRSEL_STY_ALL	(0) highlight entire row
XTRSEL_STY_CELLS	(16) highlight all cells of row
XTRSEL_STY_CELL1	(32) highlight first cell only
XTRSEL_STY_CELL1T	(64) same as XTRSEL_STY_CELL1 but highlight fills cell (not just text)

XTR_SKEY

If this field is non-null, and the current XTREE is in single-select mode, and the XTR'COLUMNACTIVE parameter is not zero, then the initial selection is established by searching the active column for the specified string (overriding the ANSWER parameter, which normally sets the initial selection). Note that:

- SKEY does not require a trailing null
- A match will succeed as long as the contents of SKEY matches the start of the data in the column (i.e. "Ab" will match "Abbey Road").
- SKEY is not cleared or set by XTREE

Update Note

XTR'SKEY was added to XTREE in A-Shell build 933 of 7 June 2005.

filidx

This optional parameter (B,4) may be specified in conjunction with file mode to create an index to the file while it is being loaded into the array. The index is dynamically allocated and kept in memory and a "handle" to the index is returned in the *filidx* parameter. The variable *filidx* cannot be used directly, nor should it be altered by the application. Its only use would be to pass it to the MIAMEX 118: Get/set stream position function, along with the desired line number (of the selected item), in order to efficiently seek to the selected item. This is mainly of use with multiple-selection file mode, since the selected items are identified only by line number via the answer array.

See the updated XTRFIL.BP sample program for an example of using *filidx*.

Note that you should only use one *filidx* variable within a program, and you should not modify its value. If you follow this guideline, A-Shell will automatically free up the memory used by the index for each new XTREE call and when the program ends.

Mouse and keyboard techniques

For controls that return selections items (rows) may be selected with a single left click. If multiple selections are allowed, use Control-left-click to select additional items, or Shift-left-click to select a range of items. Hitting the Enter key then returns this selection to the calling program. Doubleclicking with the left mouse button selects a single item and returns it. Keystrokes may also be used to select items. As you type, the control will automatically select the item which matches all of the keystrokes entered since the last pause of one to two seconds, relative to the "current" active column. (The pause restarts the selection logic.) The current column is initially the first one (or as specified by *XTR_COLUMNACTIVE*), or the last one whose header button was clicked.

Up and down arrows may also be used to move the selection bar. If exiting from the control with a navigation key (arrows, home, end, etc.) is permitted (see flags), hold down the Shift or Control keys to make the navigation key be interpreted as an exit command.

In a multi-selection list, Control+A will select all items and Control+B will deselect them.

For controls that are editable (see *XTF_EDITABLE* option in flags) , the left button is used to select an editable cell for editing, or to toggle the status of an editable checkbox. Doubleclicking on either type of cell has no effect, but doubleclicking on a non-editable cell will still act like the ENTER key.

Multi-level lists

These present some additional considerations and complications. Initially on the top level (level 0) items will be displayed. The user can click on the expand/collapse indicator at the left edge of the item to display or hide the lower level items. (See *XTR_SHOWBUTTONS*.) Sorting the column only sorts the top level (level 0) items. Dependents (lower level items) move with their respective parents, but are not themselves sorted within each group. (If the lower level items are visible at the time of the sort, this may give the impression that the sort didn't work.) When adding items to the tree control, lower level items must immediately follow their parents. (There is currently no way to insert a lower level item, chronologically out of sequence with its parent.) Since you can only have one set of column definitions, when using multiple levels, typically the higher levels may have blank columns, and you may need to interpret the contents of the columns differently.

In file mode, the level flag (as defined by the @ code in *coldef*) is returned with the selection, allowing you to interpret the result appropriately. In array mode, you get back the selected array index(es) and can refer to the original array data to determine the level of the selected item(s). Note that the array index returned will be relative to the *array* you pass, and is not affected by sorting.

Modeless operation

In addition to the additional programming required (multiple XCALL XTREE operations using different *XTR_OPCODE* values), these present the problem of how to respond to the user clicking on the listbox when some other input field has the input focus (i.e. when not inside of an XCALL XTREE operation). You can prevent this problem by disabling the control on exit (see flags *XTF_DISABLE*) or you can define a keyboard click-to-enable string in *XTR_KBDSTR*, which would be received by whatever input routine had the focus at the time the user clicked on the tree control. This would allow your application logic to take appropriate action, presumably including executing a new XCALL XTREE operation. If you neither disable the pick list when inactive, nor define a click-to-enable string in XTR_KBDSTR, then the user may have the impression of operating on the control, but your program will not be aware of it. (It could, however, retrieve the current selection(s) at some later time by executing XCALL XTREE with *XTR_OPCODE* set to 4 and *flags* set to include *XTF_NOSEL*, which would have the effect of simply returning the current selection information.)

The recommended sequence for *XTR_KBDSTR* is chr(7)+chr(250)+###. (where ### is the desired *EXITCODE* number to be returned from the INFLD call which receives the sequence.) For example, to return *EXITCODE* 175 to INFLD when the user clicks on the pick list (while an INFLD operation is active), set `XTR_KBDSTR=chr(7)+chr(250)+"175."` (Note the trailing period.)

Appending

When adding items to an existing tree control, care should be taken about the array index numbers. For example, if you initially load a tree control with a ten item array, then append another ten item array to it, the control will know that the second set of ten items should be internally numbered eleven to twenty, but it will be up to your program to realize that item fifteen is really the fifth item of the second set of ten loaded. One way to avoid this confusion is to maintain a single array which matches the contents of the tree control, even if you didn't load them all at once. In the example just cited, rather than passing two separate arrays (or worse, overwriting the original array with the second set of items), it might be better to map an array large enough for all the items, and then just add to it as you add to the tree control. The trick here is to specify the appropriate starting array index, rather than (1). For example:

```
MAP1 ARRAY(50),S,100

FOR I = 1 TO 8
   ARRAY(I) = <data>
NEXT I
! now create the control with the first 8 items...
XTR_OPCODE = 0  ! (create)
FLAGS = XTF_MODELESS + XTF_NOSEL
ADDCNT = 8
xcall XTREE, SROW, SCOL, ANSWER, ARRAY(1), ADDCNT, COLDEF, EXITCODE, &
   EROW, ECOL, FLAGS, "", MMOCLR, XTRCTL

! now load up an additional 12 items, adding them first to array
FOR I = 9 TO 20
   ARRAY(I) = <data>
```

```
NEXT I
! append these items to the existing tree control...
XTR_OPCODE = 2  ! (append)
ADDCNT = 12
! note that we pass ARRAY(9) rather than ARRAY(1) ...
xcall XTREE, SROW, SCOL, ANSWER, ARRAY(9), ADDCNT, COLDEF, EXITCODE, &
   EROW, ECOL, FLAGS, "", MMOCLR, XTRCTL
ADDCNT = ADDCNT + ADDCNT
```

Now the 20 items in our local copy of `ARRAY()` match up with the contents of the tree control, so that when we receive a selected item number via the `ANSWER` parameter, we can reference the corresponding item in our local array directly, as `ARRAY(ANSWER)`.

The initial selection(s) are determined (by the *answer* parameter) in the same way as when creating a new list, except that if you don't specify an initial selection, the default will be the first item appended. If you do specify one or more selections in the *answer* parameter, they re interpreted relative to the combined list after appending, not relative to the just the appended items. That is, setting *answer*=2 will select the second row in the table, not the second row appended.

Editable tree controls

XTREE supports both editable text columns and checkboxes. To activate this feature, set the *XTF_EDITABLE* option in the flags parameter, and define one or more columns using the E (editable text) or T (editable checkbox) options in the coldef parameter. (You must use the Advanced Syntax.) You must also map the answer in such a way to allow the updated text and/or checkboxes to be returned there.) You will probably also want to change the selection style in the *XTR_SELECTAREA* field of the xtrctl parameter so that the selection indicator does not hightlight the entire row (which makes it distracting to see the editable fields.) The XTREE control is not a true "grid", and thus it does not lend itself easily to acting like one. It probably won't, for example, work very well in an environment where every cell is editable. However, it can be quite useful in situations, like inventory or attendance, where there is a lot of data to display and only a relatively small amount to be updated. For example, consider an application where you have a list of invitees to an event, and you want to check off those that show up. You might have several display columns for each person, with a single column of checkboxes to indicate if they have arrived. In this case, there would be no need to validating the data after each check either, although you might want to allow the operator to double-click on the person's name to print a badge. Another example would be a simple inventory sheet, that lists all of the parts and has an editable text column allowing you to enter the amount on the shelf.

When the *XTF_EDITABLE* flag is set, as long as the first column is not editable, you will be able to move the selection indicator up and down with the arrow keys, and select an item by hitting ENTER. The left and right arrow keys, however, will move the "focus" to the first editable cell in the appropriate direction, after which the arrows and TAB/Shift-TAB keys will move from editable cell to editable cell. To get out editing mode, you can double click on and item (not in an editable column), or use a function key (assuming the *XTF_FKEY* bit is set in the flags parameter).

If you need to do validation of the data immediately, rather than waiting until the user performs an explicit action to terminate the XTREE editing session, you can force an immediate exit after each edited field by adding the field type code "X" to the column definition (e.g. "EX" for editable text or "TX" for editable checkboxes). XTREE will exit immediately with EXITCODE -48 as it exits from each cell whose column contains the "X" code.) It will set the *XTR'XROW* and *XTR'XCOL* parameters to indicate the cell which you exited from, and will return entire set of updated data in the ANSWER array. The application can then decide of the data is valid, perform other actions, and return to the XTREE editing mode as desired. To return to the

same cell (i.e. if it wasn't valid), you must set *XTR'XNAVCOD* = 0. Otherwise, the combination of the data passed in *XTR'NAVCOD* and *XTR'XNAVMASK* will allow XTREE to move the editing focus to the next cell, based on the navigation key used to exit the previous cell. In either case, the entire contents of the array in the answer parameter is used to update the XTREE control before proceeding with editing the next cell.

Note that you can cause individual rows (cells) within an editable text column to be ignored (skipped over) by setting the corresponding entry in the answer array to "|".

The editable option is only supported with array mode (not file mode).

See the XTRA5 sample program for a good example of editing and validation. Also see EL –CFG for a good example of editing only certain cells within an editable column.

Multiple tree controls

You may have multiple XTREE defined and displayed at once. To do so, you must specify a unique control number (0=first, 1=second, etc.) in the *XTR_CTLNO* parameter, and use the flags *XTF_MODELESS* option (so that you can exit the first list without destroying it. This kind of arrangement can be useful for allowing the user to move items between two categories, or to present in one listbox a set of possible items, and use the other to store the ones that have been chosen.

See the note next to XTR_CTLNO in the *xtrctl* table above for information on auto-assigning a control number.

Controls in container windows

You may place an XTREE control within a container window such as a Modal Dialog Box or Tab Control. In the case of dialogs, this is automatic. (That is, if a dialog box is active when the XTREE control is created, it will automatically be a child of that dialog, and it is not necessary to set the *XTR_PARENTID* parameter.) In the case of a Tab control, you must specify the control ID of the Tab control in the *XTR_PARENTID* parameter. In either case, the coordinates will be interpreted as being relative to the client area of the container window.

If you are going to open an XTREE in the middle of a screen, such that it will overlap other controls, it is highly recommended that you put it inside a dialog. Otherwise, to prevent the controls already present from "bleeding through" the XTREE, A-Shell will automatically save and delete any controls which overlap the XTREE's rectangle, and the restore them on exit. But that effect may seem a bit awkward, whereas a pop-up dialog is not only cleaner looking, but allows the user to move it around to see what is underneath. See the sample programs XTRA2 and XTRA5 for examples of XTREE controls within dialogs.

Destroying pick lists

This happens automatically in the normal modal case (i.e. when *XTF_MODELESS* not set). Otherwise, you must explicitly destroy the tree control using *XTR_OPCODE* 3, or, since it is just another control object (like buttons, static text, etc.) it may be destroyed using TAB(-1,0). If the XTREE control is within another control (such as a dialog or TAB), then you may also destroy it by using xcall MIAMEX 119 or TAB(-10,20) with opcode 4 and specifying the control ID of the parent.

Sample programs

Several sample programs (with source) are available to help you understand and use XTREE controls.

The best place to get the sample programs is from the Shared Open Source Library on the A-Shell Advanced Support Network. The latest set of sample files are there, and are discussed in a forum topic called A-Shell Shared Open Source Library.

Program	Description
TABDLG	Sample TAB control in a dialog, with an XTREE on one of the TAB pages.
XTRARY	3 column list, array mode, using the Simple Multi-Column Syntax.
XTRA2	A more sophisticated array-based example, supporting multi-level and multi-select options, and using the *coldef* Advanced Syntax to define columns with different formats, colors, etc. Also supports option to display the XTREE inside of a dialog frame.
XTRA2A	Variation of XTRA2 illustrating XTR'EXPANDALL options with a 4 level hierarchy, also merge-into and merge-out of options, row fonts.
XTRA3	An even fancier example demonstrating modeless operation, with options for multi-level and multi-selection. The program also illustrates INFLD checkboxes, buttons, TPRINT, some other GUI features, including a dialog box.
XTRA4	Simple example of using two XTREE controls at the same time.
XTRA5	Demonstrates checkboxes and editable text.
XTRFIL	File-based example. You supply the file, including the column definitions in the first row. Demonstrates both single and multiple selection options, and also the XTF_FILANS flag.

Appendix

Virtual Key Symbolic Names

The Virtual Key Symbolic Names provide an convenient notation for specifying certain control characters in control command strings (see CONTROL class *cmd* parameter). The names all start with VK_ followed by the name of the key as it relates to the keyboard. To insert these symbolic names into a cmd string, they must be surrounded by % characters (e.g. "%VK_BACK%). The names in the table below can also be altered to indicate the SHIFT or CONTROL version by inserting a "^" and/or lower case "s" after the underscore. For example, "%VK_^UP% refers to the keystroke Control-Up Arrow, "%VK_sDOWN%" refers to Shift-Down Arrow, and "%VK_^sF2%" refers to Control-Shift-F2.

Virtual Key	Meaning
VK_BACK	Backspace
VK_DECIMAL	Decimal point on numeric keypad
VK_DELETE	Delete key
VK_DOWN	Down arrow
VK_END	End key
VK_ENTER	Enter key (same as VK_RETURN)
VK_ESCAPE	Escape key
VK_HOME	Home key
VK_INSERT	Insert key
VK_LEFT	Left arrow
VK_NEXT	Next Page or Page Down Key
VK_PRIOR	Prev Page or Page Up Key
VK_RETURN	Return key (same as VK_ENTER)
VK_RIGHT	Right arrow
VK_TAB	Tab key
VK_UP	Up arrow
VK_F1 thru VK_F12	F1 thru F12 keys

Extended virtual function key codes: VK_xF###

These send chr(7) + chr(250) plus the ### (1 to 6 digits), followed by a period, which INFLD recognizes and will return as EXITCODE -###. For example, VK_xF401 will generate EXITCODE –401.

Index

A

About information, 246
Access denied, 264
Acquire (TWAIN), 150
AGF_xxx symbols, 84
AlphaPAINT, 322
AMOS command, execute, 23
AMOS_RUNSBR, 23
Any Change?, 25
Array of text, 316
ASCII to EBCDIC, 25
Ashelp.mdf, 88
AshLite, 127
Ashlog.log, 265
ATE Optimization, 93
Audio, 252
AUI_CONTROL, 30
AUI_ENVIRONMENT, 84
AUI_EVENTWAIT, 79
AUI_HTMLHelp, 92
AUI_IMAGE, 90
AUI_MENU, 86
AUI_WINDOW, 91
AutoLog, 103, 246
AutoMouse, 260
Autowidth, 120

B

Background color, 245, 246, 262
Background task, 302
Base 64 encoding, 95
BASIC errors, 111
Baud rate, 130
Beveling, 187, 244
Binary search, 293
Bitmap control, 71
Bitmaps, 97
Blanks, trimming, 296, 313
BMP files, 147
Border, 21, 194, 195, 244, 277
Box drawing, 21, 195, 244, 277, 285
BOX'xxx symbols, 277
Browser, 144
Buffer synchronization, 27
Button control, 58

C

Cache, 257
Calcuate, string length/height, 275
Calculate
  date values, 106
  hash total, 221
Calendar, 189
Cannot rename, 222
Capitalization, 319
CCYY ini file parameter, 145
Cformat, 330
CGI, 99
Character input, 22, 312
Character translation, 115
Check keyboard character, 101
Check printer, 289
Checkbox alignment, justification, 62
Checkbox control, 61
Checkboxes, 190, 192
Clickinfo, 265
Clipboard, 255
CMD file input, 226
CMDLIN, 204
Color, 118, 196, 332
  background, 245, 262
  settings, 262
Color palette, 246
Colors, 192, 246

Colors (terminal), 314
Colors, Windows, 192
COM ports, 102
Combo, 191
Combo box, 192
Combo box control, 72
Command file
  get next line, 218
  input, 225
  status, 211
Command line
  retrieving, 204
  switches, 253
Command prompt
  getting, 220
  setting, 213
COMPIL, calling, 216
Compute date values, 106
Conflicting file locks, 321
Console
  device name, 105, 137
  information, 254
context menu
  xtree, 334
Contiguous disk blocks, 98
Control-C, 28, 98, 103, 197, 291
Control-C status, 212
Control-P, 233
Controls, 30
Conversion
  ASCII - EBCDIC, 25
  date, 109
  date/time, 106, 145, 283
  date/time, 139
  floating point, 123, 142
Convert date format (INFLD), 162
Coordinate system, 34
Copy file, 221
Copy INMEMO memo, 320
CRC, 105
Create directory, 251
Creating subroutines, 10
CSV, 123
CTLOP_xxx symbols, 31
Currency field (INFLD), 158
Current directory, 221, 223
  synchronizing, 217
Cursor position, 248
Custom subroutines, 10

D

Data source (TWAIN), 151
Date control, 189
Date conversions, 106, 109, 139, 202
Date fields, 189
Date input (INFLD), 158
Date picker, 189
Date picker control, 74
Date shortcuts (INFLD), 162
Date time control, 189
Date, INFLD, 189
Date/time conversion, 145, 283
Decimal point (INFLD), 160
Decimal, PPN values, 323
Default file extension, 223
Delete memory module, 258, 261
Device definitions, retrieving, 218
Device information, 132
Dialog box, 279
Directory
  create, 224, 251
  delete, 224
  scanning, 220
  separator character, 220
Directory, system, 223
Disable printing, 121
D-ISAM, 27
Disk space, 98
Display, AlphaPAINT, 322
Divide by zero, 27
DLL location, 66
DO file parameters, 239, 240
Double precision, 123
DPRINT parent control, 274
Drawing, 21, 194, 195, 244, 277, 285
dump control spreadsheet, 55

E

EBCDIC to ASCII, 25
Echo, 110, 282
Echo status, 213
Edit control, 71
Edit controls, 187
Email, 95, 110, 262
Encoding, base 64, 95
Encrypted files, 113
Environment variable, 223
Error messages, 242
  printing, 222
Error trapping, 111

Ersatz info, getting, 215
Ethernet controller, 135
Eventwait, 79
EVW_xxx symbols, 81
Exclusive file locks, 321
Exclusive locks, 125
Execute AMOS command, 23
Execute Host Command, 142
Exit A-Shell, 211
Expand file, 268
Expiration, 246
Exporting data, 123
EZ-SPOOL, 115

F

Field lengths, 192, 193
FIFO, 123
File, 338
  privileges, 248
File access in SBX, 243
File extension, set default, 223
File locking, 27, 125, 321
File position, 263
File privileges, 219
File size, getting, 294
File spec conversion, 211
File stats, 267
File time, formating, 268
File, expand, 268
File, line count, 274
Find matching file, 219, 220
Fixed pitch, 192, 193
FKEYWAIT, 241
Floating point conversion, 123, 142
Floating window, 279
Flow control, 130
FLSET, 243
Flush output buffer, 243
Folding, 319
Fontface, 30, 41
Fonts, 192, 193
Form names, 117
Free disk space, 98
Freud, Sigmund, 319
FSPEC, 211
Function key, 22
  emulate, 33
  input sequence, 21
  sequence completion wait, 241
  translations, 21, 25, 130, 138, 232
Function key, virtual, 188

G

Get environment variable, 223
Global DO file parameters, 240
GOP_xxx symbols, 229
GOP2_xxx symbols, 230
Grid, 34
Groupbox control, 58

H

Hash, 224
Hash compability mode, 224
Hash total, 221
Height, 21
Help button, 280
Hex mode, 213, 214
HexaDecade dates (INFLD), 162
HHOP_xxx symbols, 92
Hidden controls, 32
HKEY_xxx symbols, 251
HLPIDX, 187, 189
Horizontal scrolling, 184

I

Icon buttons, 60
Icon control, 65
Identify device, 105
IDX lock, 28
IEEE floating point, 123, 142
Images in dialogs, 149
Imaging, 147
Importing data, 142
Index files, searching, 292
INFLD, 189
INFLD.DEF, 185
INI.CLR, 118, 262, 287
INMEMO, copy or move, 320
Input
  buffer, 101
  character, 21, 102, 129, 131, 138, 312
  file channel, 130
  stream, 126
Input, characters, 318
Input, type-ahead, 318
inSight, 318
Instances of A-Shell, 23
INSTR, 281
Insufficient privileges, 121

Inter-task communications, 198
INXCTL, 188
IP address, 266
ISAM IDX lock, 26
ISAMPLUS files, 27
ISAMPLUS version, 223

J

Job control block, 133, 236, 293
Job information, 132, 201, 288, 323
Job name, 133, 201, 236, 288
Job table record, zapping, 225
Jobname, unique filename, 314
JOBTBL.SYS, 236, 237
JPG files, 147
Julian date, 107, 109, 202
Julian date format (INFLD), 163
Justify field (INFLD), 160, 161

K

Keepalive, 187
Keyboard input, 130, 138
Keys, sort, 96

L

Landscape, 116
Language, 235
Language definition, 25, 140
Large bit fields, 97
Large sort, 96
Last line number, retrieving, 274
Launch windows app, 250
LDF, 140
Lead-in character, 187
Leading blanks, trimming, 313
License info, 246
License, ISAM, 223
Line draw, 21, 57, 244, 277
Line input mode, 130
Line insert, 194
Line numbers, retrieving, 274
Lines in file, count, 274
Linux version, 255
LITMSG, 235
Load image, 148
Load memory module, 257
Local copy, 28
Locking, 125, 321
Locking daemon, 27
Logical device, 108
Logical memo, 320
Login directory, 211, 217, 221
Login name, 137
LOKSER, 28, 225

M

MAC address, 135
Mail, 262
Make directory, 251
MAP, memory, 256
MAP.LIT, 256
MAPI, 262
Matching file or directory, 219, 220
Maximum modules, 257
MBF_ALTPOS, 60
MBF_xxx symbols, 32, 87
MBST_xxx symbols, 32
Media Access Control, 135
Medium sort, 96
MEM device, 26
Memo file(s), 320
Memory mapping, 27
Memory module, save, 258
Memory partition, 256, 261
Memory, user, 257
menu
    xtree popup, 334
Menu, 22, 86
Menu operations (Windows), 86
Message box, 279
Message box title, 193
Messages, 235
Messages (between jobs), 198
Millirows, 39
MIME, 95
Minus sign (INFLD), 160
MME_xxx symbols, 197
MMO_xxx symbols, 194
MMX_xxx symbols, 197
Modal dialog box, 75
Month, 107
Mouse, 260
Mouse click reporting, 265
Mouse click sequences, 36
Mouse cursor reporting, 187
MSBOXX, 303
MSGBOX, 280
Multi-line edit control, 72

N

Named pipes, 123
Network disconnect, 187
NFS, 27
Nodes, 246
Non-printable characters, removing, 313
Numeric formatting codes (INFLD), 160

O

Octal mode, 213, 214
Octal, PPN values, 323
OCVT, 214
OFN_xxx symbols, 249
Open file dialog, 248
Operating system version, 255
OPTIONS, 229, 230
Orientation, 116
OT_xxx symbols, 214
Output buffer, 243
Ownership, file, 219, 248

P

Packets, 154
Paint, 322
Paper orientation, 116
Parent control (TPRINT, DPRINT), 274
Parse strings, 297
Partition, 257, 261
Partitioning, 98
PCKLST, 324
PCX files, 147
PFK, load, 232, 233
Pick lists, 285
PID, 255
Pipes, 123, 124, 226
Pixel coordinates, 39
Play a sound, 252
PolyShell, 201, 246
PolyTrack, 288
Pop-up lists, 285
Pop-up utilities, 186
Pop-up window, 279
Port, 102, 264
Port filters, 115
Portrait, 116
PPN, 109, 132, 133, 136, 203, 214, 217, 236, 242, 288, 323
PPN, octal vs. decimal, 323
Pre-compiler, 217
Print, 115
  file viewer, 122
  individual pages, 122
  preview, 115, 121
  queue, 115
Print files, 314
Print filter
  EMAILP, 110
  GDIPRT, 127
  HTMLP, 144
Printer
  field, 118
  selection, 115, 264
  status, 289
  Windows, 264
Privileges, 219, 228, 248, 317
Privileges, read, 264
Process ID, 255
Progress bar control, 76
Proportional fonts, 192, 193
Pseudo, 188
Pseudo function keys, 36, 188
Public, John Q., 319

Q

Queue block, reading, 222
Queue blocks, zapping, 225
Queue handling, 233, 234
Quicksort, 95

R

Radio button control, 64
Radio buttons, 33, 190
RANDOM'FORCED, 27
Raw characters, 129
Read privileges, error, 264
Read'only flag, 27
Record locking, 125, 321
Record selection logic, 205
Rectangles, 57
Rectangular text area, 316
Registry, 251
REGOP_xxx symbols, 269
Restricting program access, 264
RGKEY_xxx symbols, 270
right click menu, 334
Right justify field (INFLD), 161
Rounding, 266
Row and column determination, 314

Rows and columns, 39
RUNSBR, 23

S

SBX, 10
Scaling, 192, 193
Scanning directories, 219, 220
Screen
  picture, 233
  preview, 115
  snapshots, 186
Screen, AlphaPAINT, 322
Security, 113, 135
Selecting records, 205
Selection menu, 22
Separator character, 220
Sequential file, 290
Sequential search, 293
Sequential sort, 95
Serial communications, 102
Serial number, 246
Serial number, ISAM, 223
Serial port, 102, 110, 130, 282
Server process, 305, 306
SETRO, 26
Shell execute, 250
Show window, 239
Signals, 226, 228
Single precision, 123
Sink / unsink Windows box, 266
Sleeping job, 316
Small sort, 96
Sockets, 305, 306, 308
Sort, 95
Sort keys, 96
Sorting arrays, 295
Sound file, 252
Spaces, trimming, 296, 313
SPEC, 211
Speed Optimized Interface, 131
Spool, 115
spreadsheet (of controls), 55
SSD chip, 135
Startup (Windows), 266
Static text control, 56
Stats, file, 267
stdout, 99
Stream buffer, 243
Stream position, 263
string replacement, 99
String, calculate length/height, 275
Strings, parsing, 297
STRTOK, 297
Subroutines, writing, 10
substring replacment, 99
Swap, 303, 316
Swap wait time, 253
Symbolic names, 349
Synchronized write, 27
Synchronizing directories, 217
SYSLOG.LIT, 281
System directory, 223
System message file, 265
System time, 237
SYSTEM.SPL, 119

T

Tab control, 77
Tabs, removing, 296
TCP/IP, 305, 306
TCPCLI, 308
TCPSRV, 308
TCPXFLG'xxx symbols, 310
TCS_xxx symbols, 77
Telnet server, 238
Temp directory, 28
Template, web form, 100
Temporary files, 314
Terminal characteristics, 314
Terminal driver name, 133, 201, 236, 288
Terminal echo, 110, 282
Terminal ID, 105
Terminate directory search, 220
Text box, 316
Text file viewer, 122
Text object, 56
Thousands separator (INFLD), 160
TIF files, 147
tilde, embedding in title, 330
Time, 237
Time control, 189
Time field (INFLD), 163
Time fields, 189
Time picker control, 75
Time/date conversion, 145, 283
Timeout, 187
Timeout (EZTYP), 122
Title bar, Windows, 242
Tooltip, 30, 42
Topics, HTML, 92

TPRINT parent control, 274
TRACE, 231, 232
Tracker, 215, 277, 303
Trailing blanks, trimming, 296, 313
TROP_xxx symbols, 231
TWAIN, 151
Type ahead, 291
Type-ahead buffer, 318

U

UMASK, 234
UNIX version, 255
Upper case (INFLD), 162, 163
URL, link to, 250
User information, 120, 133, 137, 288
User memory, 26, 256, 257, 258, 261
User name, 133, 288
USRMEM'xxx symbols, 257

V

Version
    A-Shell, 215, 246
    OS, 255
Virtual function keys, 188
Virtual Key Symbolic Codes, 37
Virtual keys, 349
VUE, calling, 216

W

Wait for Event, 79
Wake job, 316
WAV files, 252
Web pages, 100
Web server, 99
Week, 107
Width, 21
Wildcard file locks, 321
Window
    display options, 239
    title, 242
Windows colors, 192
Windows edit controls, 187
Windows version, 255
Workstation ID, 105
Write operations, 27
Write to sequential file, 290
WRITECD, 123
WRITETD, 123

X

XP themes, 93
XPAINT, 322
XPPN, 323
XTF_xxx symbols, 336
XTROP_xxx symbols, 340
XTRSEL_xxx symbols, 342

Z

Zap a user, 225
Zero fill (INFLD), 161